CIP2017-06-18
Multiple Graphs

oClG 4 Presentation, 2017-08-17

Stefan Plantikow, Neo4j
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 at this time, nor does this presentation represent any commitment that Neo Technology is going to provide such features in the future.
Outline

Motivation

Status

Model

Language Changes

Example

Extras and Variations

Next Steps
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)
Graph Transformation

FROM GRAPH AT "...
MATCH (a)→()→(b)
RETURN GRAPH (a)→(b)
Support for multiple graphs impacts...

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

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

**Language Model** 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
Status

- Ongoing design effort, public since oCIM 1
- CIP is undergoing rapid revision
- Readable draft detailing the overall direction
- Goal:
  - Consistent, polished draft for next oCIG
  - Final CIP for oCIM 3
- Core is being implemented in Cypher for Apache Spark (CAPS)
Physical Model
Physical model today: Single Graph

Entity = Node | Relationship
Physical model: Graph Space
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)
Property Graph Model
Property Graph Model: Graphs as entity containers

- Graphs are 1st class entities with an address (Graph URL)
- Metadata may be added later (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
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.
Language Model
Language Model: CYPHER 2017

Cypher 2017
Language Model: CYPHER 2018

G1  G2  G3  T1

G1  G2  G4  T2

Cypher 2018
Basic language changes

- Selecting which graph to match from
- Creating and updating graphs
- Selecting which graph to write to
Selecting graph to query from

Assume a set of named argument graphs for the query: foo, bar, baz
Which graph is queried by MATCH?

FROM GRAPH cities
MATCH (city:City)-[:IN]->(:Country {name: "Germany"})
FROM GRAPH people
MATCH (person)-[:LIVES_IN]->(city)
RETURN person ORDER BY person.age LIMIT 1
Creating graphs

Graphs are created by giving a more complex "graph definition" in FROM

- **GRAPH foo**: Existing graph foo
- **NEW GRAPH bar**: New empty graph bar
- **COPY GRAPH foo TO baz**: Copy existing graph

Graphs may be stored at a graph URL, a filesystem path, or under a name from a database catalog

- **GRAPH foo AT "url"**: Load graph from "url"
- **NEW GRAPH foo AT "url"**: Create graph at "url"
Selecting graph to write to

Assume a set of named argument graphs for the query: foo, bar, baz
Which graph is written to by CREATE, MERGE, DELETE?

FROM GRAPH people
MATCH (alice)-[:KNOWS]->(bob)-[:KNOWS]->(charlie)
WHERE (bob)-[:LOVES]->(alice)<-[:LOVES]-(charlie)
AND (bob)<-[:LOVES]-(alice)-[:LOVES]->(charlie)
INTO GRAPH drama
CREATE (t:LoveTriangle)<-[:IN]-(alice),
     (t)<-[:IN]-(bob), (t)<-[:IN]-(charlie)
RETURN DISTINCT alice.name
Clause semantics for multiple graphs

- [OPTIONAL] MATCH: read from source graph
- Expression: read from source graph
- CREATE:
  - add bound nodes to target graph
  - create unbound nodes in target graph
- MERGE: read from source graph, write to target graph
- SET/REMOVE: update entities in target graph
- DELETE: delete entity in target graph
Transforming Graphs
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></td>
</tr>
<tr>
<td></td>
<td>(#1)--&gt;(#3)</td>
<td>a: #1, b: #3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(#3)--&gt;(#2)</td>
<td>a: #3, b: #2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(#3)--&gt;(#4)</td>
<td>a: #3, b: #4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(#4)--&gt;(#2)</td>
<td>a: #4, b: #2</td>
<td></td>
</tr>
</tbody>
</table>
Graphs from tables

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

WITH a, r, b
RETURN
GRAPH (a)-[r]->(b)
AS foo
Transforming graphs

Using the update language is verbose, declarative graph transformation avoids it

MATCH (a:Person)-[:FRIEND*2]->(b:Person)
RETURN GRAPH (a)-[:FOAF]->(b) AS foo

can be combined with previous syntax

MATCH (a:Person)-[:FRIEND*2]->(b:Person)
WITH GRAPH (a)-[:FOAF]->(b)
// Similar to (c)-[:FRIEND*4]->(d)
MATCH (c)-[:FOAF]->(d)
RETURN *

2017 Stefan Plantikow, Neo4j
Projecting and managing graphs
Projecting graphs

Graphs are top-level named objects in a query. Graphs are aliased, dropped, introduced in WITH/RETURN like regular variables.

```
RETURN GRAPH foo
RETURN name, age, GRAPHS foo, bar, baz

WITH GRAPHS *
WITH GRAPHS *, NEW GRAPH myNewGraph AT "url"
WITH GRAPHS foo, bar, COPY GRAPH foo TO baz
```
Projecting values and graphs

RETURN a, b, c, GRAPH foo, bar
RETURN a, b, c

WITH a, b, c, GRAPH foo, bar
WITH a, b, c
WITH GRAPH ...

Project given fields, project given graphs
Project given fields, discard all graphs

Project given fields, project given graphs
Project given fields, project all graphs
Discard all fields, project given graphs
Projection resets source and target graphs

- **WITH ... GRAPH foo**
  'foo' becomes source and target graph

- **WITH ... GRAPHS foo, bar, baz**
  Need to re-establish source and target graph (!)

- If no source has been specified, it is chosen to be the **ambient graph**
  This is expected to be provided by the session
  (and could be empty).
Changing source and target during projection

**WITH GRAPHS** may be used as a query signature for query composition

Having to reset source and target using **FROM/INTO** is verbose

Suggested shorthand

**WITH GRAPHS** foo, bar >> source=bar

**WITH GRAPHS** foo, >> baz target=baz

**WITH GRAPHS** foo >> bar source=foo, target=bar
Managing persistent graphs

- COPY GRAPH foo TO "new-url"
- MOVE GRAPH foo TO "new-url"
- DELETE GRAPH foo
Multiple Graphs Cypher Summary

- Cypher supports named graphs as input and output to a query
- Refer to graphs using DSL of graph definitions
- Clauses work using specified source and target graphs
- Allow use of both DML and graph transformation for creating new graphs
- Cypher becomes graph compositional
Example 1: Data integration

Graph 1:

Actors, Films, Cities and People
Example 1: Data integration

Graph 2: Events and Cities
Example 1: Creating a new graph (Step 1/3)

```
FROM GRAPH ActorsFilmsCities AT 'graph://actors_films_cities...

MATCH (p:Person)-[:BORN_IN]->(c:City)
INTO NEW GRAPH PersonCityEvents

   (p:Person {name: p.name, YOB: p.YOB})
MERGE (c:City {name: c.name})
MERGE (p)-[:BORN_IN]->(c)

RETURN GRAPH PersonCityEvents
```
WITH GRAPH PersonCityEvents
FROM GRAPH Events AT 'graph://events...'
MATCH (c)<-[[:IN_CITY]]-(e)-[:IN_YEAR]->(y),
    (e)-[:IS_A]->(et {value: 'Criminal Event'});

// Do matches for all other event types:
// Public Event, War Event....
...

INTO GRAPH PersonCityEvents
MERGE (c:City {name: c.value})
MERGE (e {title: e.value, year: y.value})
MERGE (e)-[:HAPPENED_IN]->(c)
SET e:WarEvent

// Do for all remaining event types
...

RETURN count(*) AS matches
WITH GRAPH PersonCityEvents
MATCH (ce:CriminalEvent)-[:HAPPENED_IN]->(c:City)

INTO NEW GRAPH Temp-PersonCityCrimes
MERGE (p:Person {name: p.name, YOB: p.YOB})
MERGE (c:City {name: c.name})
MERGE (ce:CriminalEvent {title: ce.title, year: ce.year})
MERGE (p)-[[:BORN_IN]]-(c)
MERGE (ce)-[:HAPPENED_IN]-(c)
RETURN GRAPH Temp-PersonCityCrimes
Example 2: Aggregating data (‘roll-up’), and returning a graph and a table

Graph:

Stores, Regions, Products and Orders
Example 2: Aggregating the data

FROM GRAPH SalesDetail AT ‘graph://...’
MATCH (p:Product)-[rel:IN]->(:Order)<-[HAS]-(s:Store)-[:IN]->(r:Region)
WITH reg.name AS regionName,
    s.code AS storeCode,
    p.code AS productCode,
    sum(rel.soldPrice * rel.numItemsSold)
AS storeProductTotal

<table>
<thead>
<tr>
<th>regionName</th>
<th>storeCode</th>
<th>productCode</th>
<th>storeProductTotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>APAC</td>
<td>AC-888</td>
<td>PEN-1</td>
<td>20.00</td>
</tr>
<tr>
<td>APAC</td>
<td>AC-888</td>
<td>TOY-1</td>
<td>45.00</td>
</tr>
<tr>
<td>EMEA</td>
<td>LK-709</td>
<td>BOOK-2</td>
<td>10.00</td>
</tr>
<tr>
<td>EMEA</td>
<td>LK-709</td>
<td>TOY-1</td>
<td>40.00</td>
</tr>
<tr>
<td>EMEA</td>
<td>LK-709</td>
<td>BOOK-5</td>
<td>15.00</td>
</tr>
<tr>
<td>EMEA</td>
<td>WW-531</td>
<td>BOOK-5</td>
<td>18.00</td>
</tr>
<tr>
<td>EMEA</td>
<td>WW-531</td>
<td>BULB-2</td>
<td>190.00</td>
</tr>
<tr>
<td>EMEA</td>
<td>WW-531</td>
<td>PC-1</td>
<td>440.00</td>
</tr>
</tbody>
</table>
Example 2: Returning the graph and table

RETURN regionName,
storeCode,
sum(storeProductTotal) AS totalStoreSales
GRAPH SalesSummary // not shown

<table>
<thead>
<tr>
<th>regionName</th>
<th>storeCode</th>
<th>totalStoreSales</th>
</tr>
</thead>
<tbody>
<tr>
<td>APAC</td>
<td>AC-888</td>
<td>65.00</td>
</tr>
<tr>
<td>EMEA</td>
<td>LK-709</td>
<td>65.00</td>
</tr>
<tr>
<td>EMEA</td>
<td>WW-531</td>
<td>648.00</td>
</tr>
</tbody>
</table>
Extras + Variations
Suggestion: USE/FROM/INTO instead of FROM/INTO

**USE**  (New) Set both

**FROM**  Set source graph only

**INTO**  Set target graph only
Suggestion: SNAPSHOT & Copy patterns

Graphs may be introduced as snapshot graphs:

```
COPY GRAPH foo AS SNAPSHOT bar
```

Effect: Added entities will always be copied

Also achievable in regular graphs using copy patterns:

```
CREATE (=p)
```
Suggestion: Graph labels and properties

SET GRAPH foo.prop = 42
REMOVE GRAPH foo.prop

SET GRAPH foo:SocialNetwork
REMOVE GRAPH foo:SocialNetwork

How to query this?
Suggestion: Graphlets

- Bind individual matches to tiny subgraphs
- Enables composition: &intersect, |union, -difference, !complement (against source)
- Useful for graph transformation
- Syntax idea:

```
MATCH (a:Person),
    <g1 (a)-[:KNOWS]->(...)>,
    <g2 (a)-[:LIVES_IN]->(...)>
RETURN GRAPH g1-g2 AS g
```
Next steps

Continued work on the CIP

Feedback wanted

Final draft for oCIM 3