Towards GQL

Composable Graph Queries and Multiple Named Graphs

in Cypher for Apache Spark

Stefan Plantikow, Neo4j

Welcome

Stefan Plantikow

Product Manager / Cypher Specification Editor
Former Lead Cypher for Apache Spark

Neo4j

CAPS Team @ Neo4j

Mats Rydberg
Martin Junghanns
Max Kiessling
Philip Stutz

LANGSTAR Team @ Neo4j

Alastair Green
Tobias Lindaaker
Petra Selmer

Agenda

Cypher overview

Multiple graphs in Cypher (CIP2017-06-18 and related)

Cypher products with support for multiple graphs

- Cypher for Apache Spark
- Demo

The future: GQL

The Cypher query language

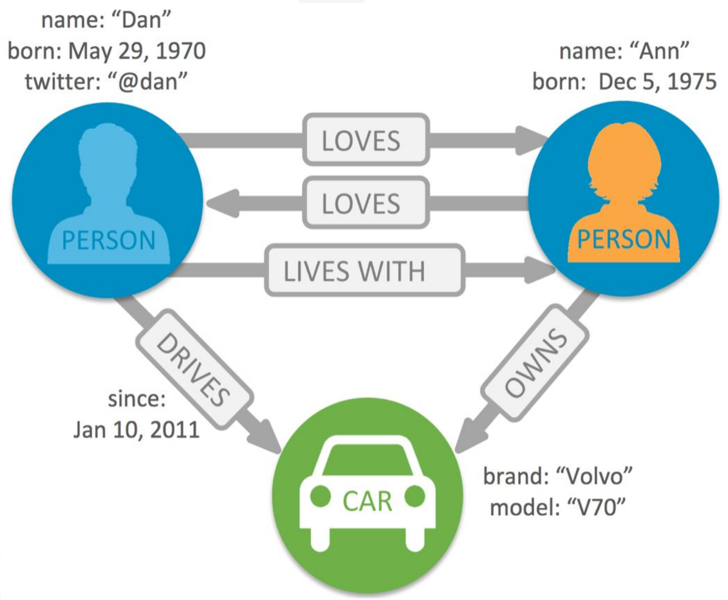
Cypher

- Original declarative query language for the property graph model
- Invented at Neo4j by A. Taylor in 2011
- Edge-isomorphism by default

- Continuous evolution: DML, Labels, DDL, MG, Path patterns
- Inspiration: PGQL, G-CORE, SQL PG Ad-Hoc => GQL
- Cypher today: Formal semantics (SIGMOD), Multiple graphs (This talk)

- Next stage: GQL

Property graph



Node (vertex)

- Represents an entity within the graph
- Has zero or more *labels*
- Has zero or more *properties*

Relationship (edge)

- Adds structure to the graph
- Has one *type*
- Has zero or more *properties*
- Relates nodes by *type* and *direction*

Must have a start and an end node

Properties

- Key-value map associated with nodes and relationships
- Represents the data: e.g. name, age, weight etc
- *String* key; typed value *(string, number, bool, list)*

Cypher principles

graph patterns as a fundamental language construct for

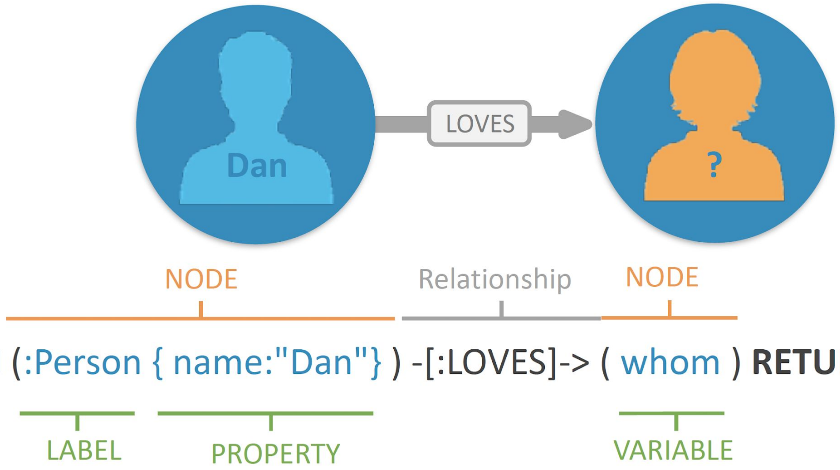
- graph matching
- graph updates
- graph construction
- constraint and index declaration

SQL-inspired (clauses, subclauses, expressions, ternary logic, `NULL`)

linear (top-down) **composition of clauses** (lateral join / flat map)

built-in **structured data types**: lists, maps

Searching for (matching) graph patterns



```
MATCH (:Person { name:"Dan"} ) -[:LOVES]-> ( whom ) RETURN whom
```

- Recursive queries
- Variable-length relationship chains

- Full RPQs (proposal)
- Path-binding queries

DQL: reading data

Linear Top-Down Flow

Implicit Input: a property graph

```
// Pattern description (ASCII art)
MATCH p=(me:Person)-[:FRIEND*]->(friend),
        (me)-[:FROM]->(:City)<-[:FROM]-(friend)

// Filtering with predicates
WHERE me.name = ‘Frank Black’AND friend.age > me.age

// Projection of expressions
RETURN toUpper(friend.name) AS name, friend.title AS title, p

// Order results
ORDER BY name, title DESC
```

Output: a table

DML: Creating and updating data

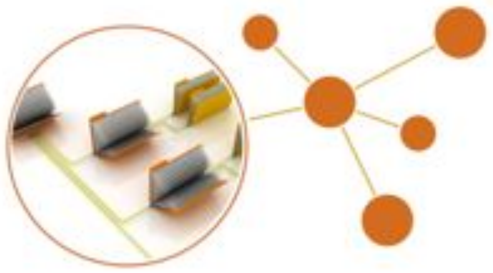
```
// Data creation and manipulation
CREATE (you:Person)
SET you.name = ‘Jill Brown’
CREATE (you)-[:FRIEND]->(me)


// Either match existing entities or create new entities.
// Bind in either case
MERGE (p:Person {name: ‘Bob Smith’})
RETURN p.created, p.updated
```

Use cases in industry

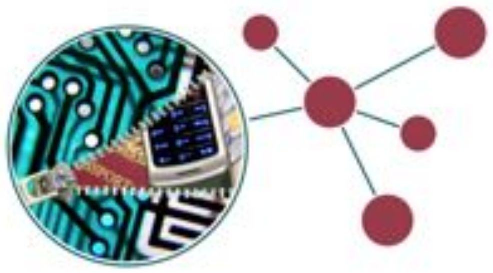
Impact Analysis

Logistics and Routing

Recommendations



Access Control



Fraud Analysis

Social Network

Query Composition requires Multiple Graphs

Motivating multiple graphs & query composition

Combining and transforming graphs
from **multiple sources**

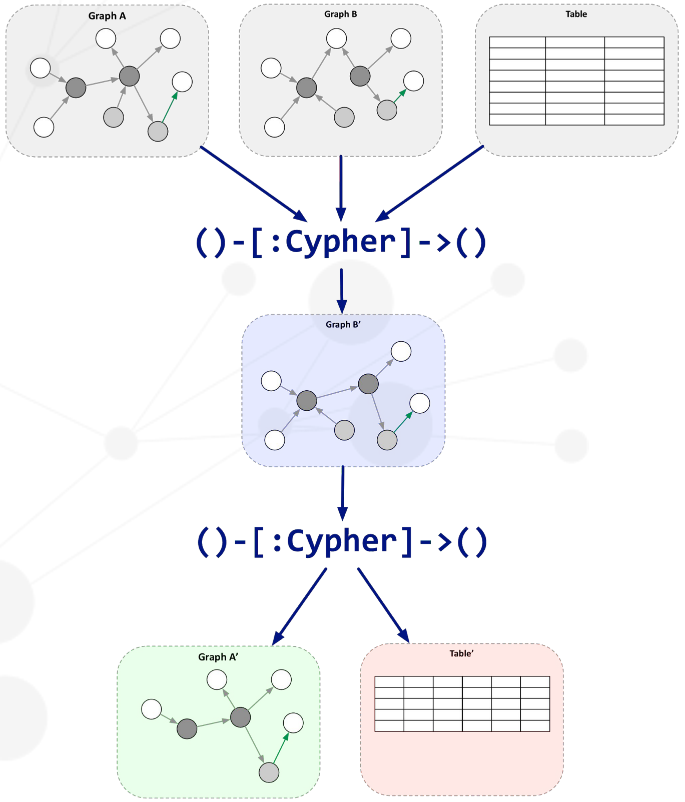
Versioning, snapshotting, computing difference graphs

Graph views e.g. for access control

Shaping and **integrating** heterogeneous graph data

The output of one query is used as the input to another

- Organize a query into multiple parts
- Extract parts of a query to a view for re-use
- Replace parts of a query without affecting other parts
- Build complex workflows programmatically



Key design choices

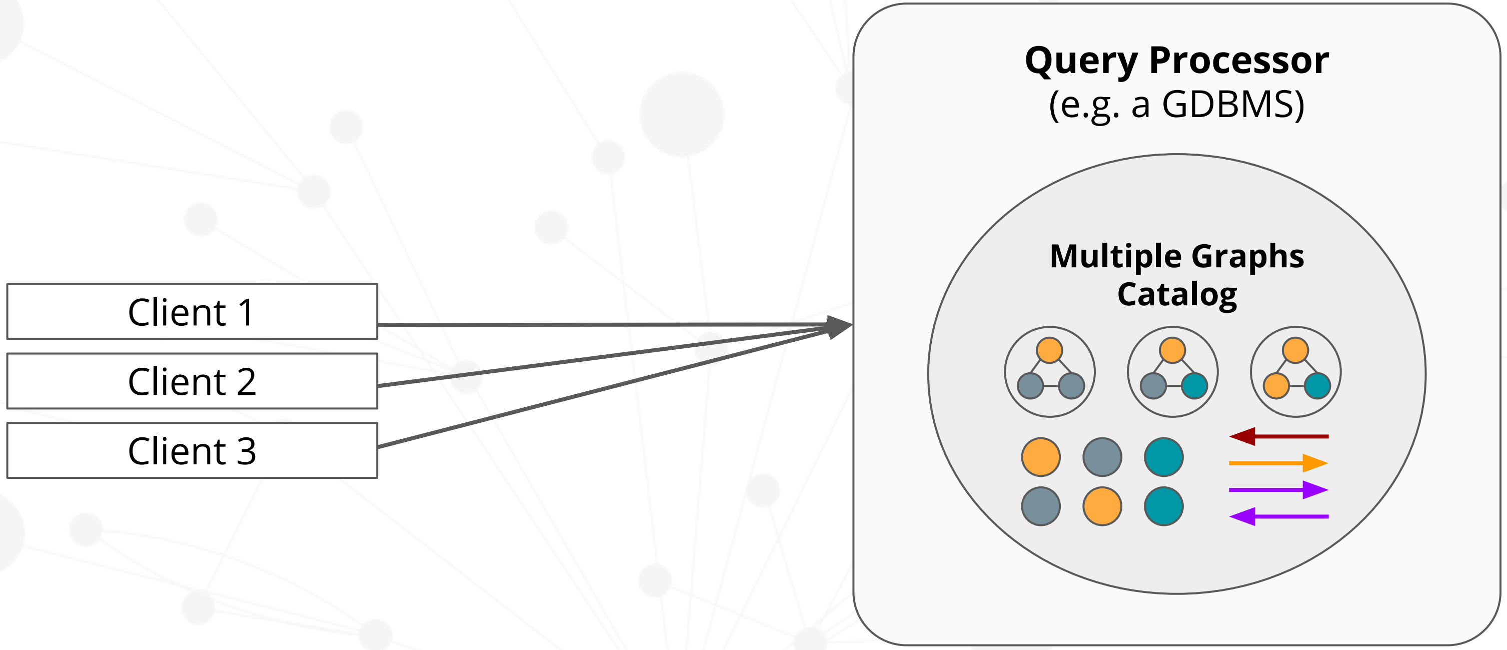
Global graph catalog

Clauses operate within the context of a single working graph

Graph construction projects updatable views using DML syntax

Successful queries return either a graph, or a table

Cypher: multiple graphs model



Working graph interaction

```
// Set the working graph to the graph foo in the catalog for reading
FROM foo
...

// Set the working graph to the graph foo in the catalog for updating
UPDATE foo
...

// Construct new working graph
CONSTRUCT
...
// Return current working graph as a result
RETURN GRAPH
```

Example: Reading from multiple graphs

Which friends to invite to my next dinner party?

```
FROM social_graph
MATCH (a:Person {ssn: $ssn})-[:FRIEND]-(friend)
FROM salary_graph
MATCH (f:Employee {ssn: friend.ssn})<-[:INCOME]-(event)
WHERE $startDate < event.date < $endDate
RETURN friend.name, sum(event.amount) as incomes
```

Graph construction

Graph construction dynamically constructs a **new working graph**

- for querying, storing in the catalog, later updating
- using entities from other graphs (this is called replication)

Simple example

```
MATCH (a)-[:KNOWS {from: "Berlin"}]->(b)
CONSTRUCT
MERGE (a), (b) // replication, aka "shared entities"
CREATE (a)-[:MET_IN_BERLIN]->(b)
RETURN GRAPH
```

Replicating entities

Take an **original entity** and create a **representative replica** in the constructed graph

```
MATCH (a)
CONSTRUCT
MERGE (a)
```

Replicating the **same original entity** multiple times still only creates a **single replica.**

```
MATCH (root)-[:PARENT_OF*]->(child)
CONSTRUCT
MERGE (root), (child)
CREATE (root)-[:ANCESTOR_OF]->(child)
```

Variations: Replicating relationships replicates start and end nodes, `MERGE GRAPH`, `MERGE PATH`

Replication is useful for updatable views and graph union. It relies on provenance tracking.

Provenance tracking aka entity sharing

- **Data model**: Entities belong to one and only one graph
  `Node #1 in Graph #1`

- **Provenance graph**: Tracks entities across graph construction
  `Node #2 in Graph #2 is a replica of Node #1 in Graph #1`

- **Entity values:** References to a replica group with the same root
  `n references Node #1 in Graph #1 and all of its replicas (e.g. Node #2 in Graph #2)`

-
  ```
  graph(n)    - Graph of root, e.g. Graph #1
  id(n)       - id of root, e.g. #1
  a=b         - graph(a) = graph(b) && id(a) = id (b)
  ```

Updatable views

```
CONSTRUCT
// build the view (track provenance information)
...
UPDATE GRAPH
// update entities in the view (use provenance information)
...
```

Graph operations

Entities are always replicated

```
CONSTRUCT
...
RETURN GRAPH
UNION | INTERSECT | EXCEPT | UNION ALL | ...
CONSTRUCT
...
RETURN GRAPH
```

Catalog side-effects

```
CREATE GRAPH foo  // Create new graph 'foo'
DELETE GRAPH foo  // Delete graph 'foo'

COPY foo TO bar   // Copy graph 'foo' with schema
RENAME foo TO bar // Rename graph 'foo' to 'bar'
TRUNCATE foo      // Remove data but keep schema in 'foo'


// Extensions
ALIAS foo TO bar      // Aliasing
... GRAPH foo TO bar  // Error if 'foo' is not a graph
... GRAPH TO bar      // Use working graph
```

Design summary

- select-construct-return
  - essence of composition: what's operated on is what's produced
- working graph serves as operational context
  - preserve existing mental model of Cypher: implicit graph + driving table
  - fits nested subqueries (outer working graph => initial working graph)
- graph construction uses DML
  - leverage knowledge of existing DML semantics for users
  - allows negative graph construction (`MERGE GRAPH` + `DELETE`)
  - future work: graph aggregation
- graphs track provenance
  - essential for graph operations: entities from returned graphs can be related to base data

Cypher for Apache Spark

Cypher implementations

Industry

- SAP HANA Graph
- Redis Graph
- Agens Graph
- Neo4j
- Memgraph
- **Cypher for Apache Spark (This talk)**
- **Cypher for Gremlin (Not this talk but please ask)**

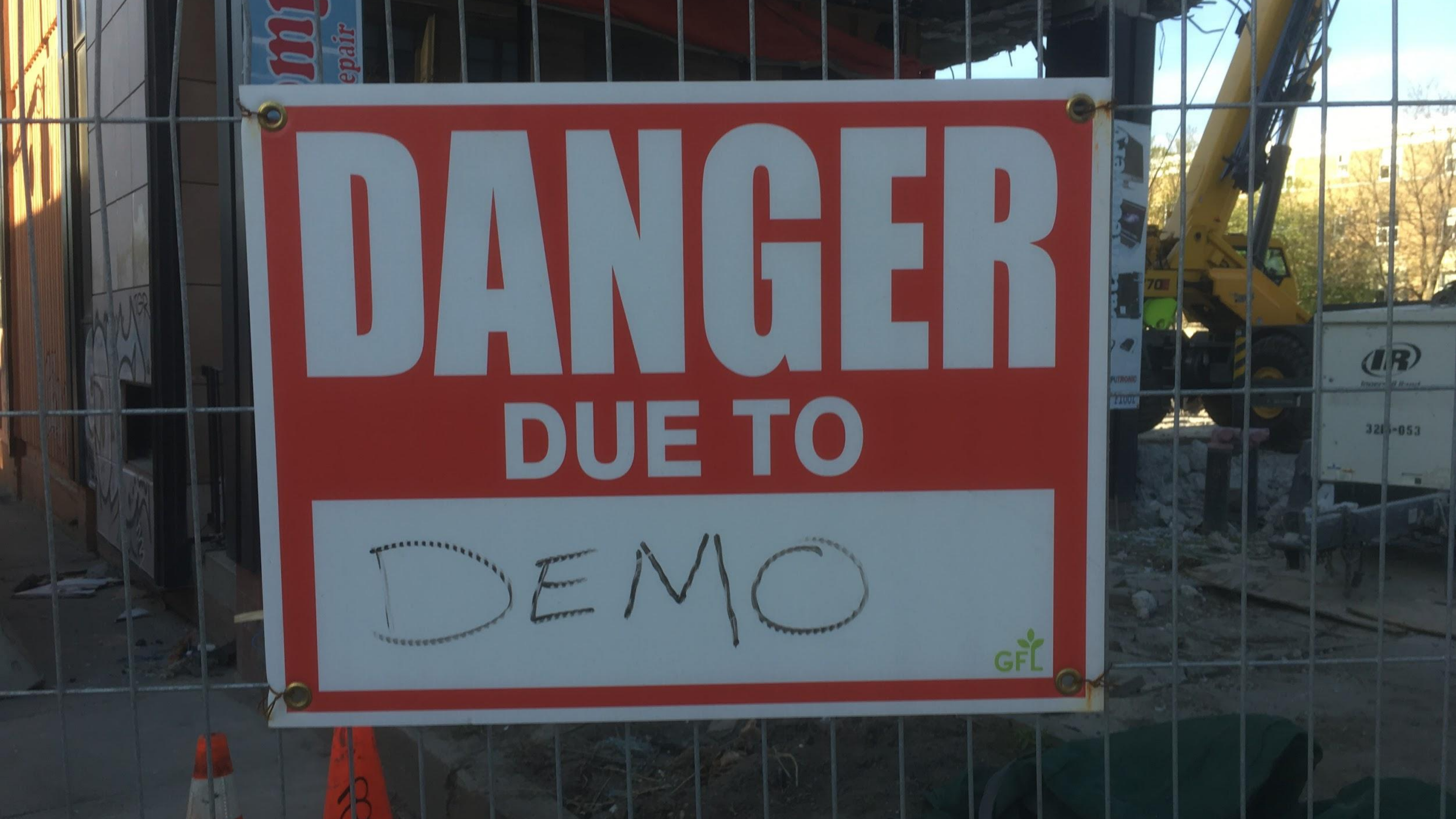
Research

- Gradoop (*Distributed Graph Analytics on Apache Flink*): U. of Leipzig
- ingraph (*Incremental evaluation of Cypher queries*): U. of Budapest
- Graphflow (*Supporting continuous queries and triggers*): U. of Waterloo

Cypher for Apache Spark

- Full Cypher implementation for Apache Spark
  - Neo4j Cypher Frontend
  - Custom IR and query planner
  - Target: DataFrame API

- Programmatic API (similar to SparkSQL)
- Multiple data sources
- Commercial product: Neo4j Morpheus (Big data integration)


DANGER
DUE TO
DEMO
GFL

GQL

gql.today

The GQL manifesto:
Avoid market confusion and divergence - fuse Cypher, PGQL, G-Core into GQL

Please join the cause and sign the GQL manifesto

Next step proposal: Jointly work on feature comparison
(similar to short comparison document on gql.today site but with more detail)

Take it from there

Let's discuss more

Summary

Cypher is the defacto standard property graph query language with >=10 implementations

Multiple graphs are necessary for query composition
(More information in CIP2017-06-18 and oCIM 4 slides)

Multiple graphs Cypher is available in CAPS now

Cypher is evolving to be the next generation query language for graphs: GQL

Thank you!

If you have a questions, a research topic, or would like another demo, please come and speak to us!