Cypher

Cypher overview

Cypher is a family of query languages for Property Graphs:

- Proprietary query language of the Neo4j graph database
- Subset supported by other tools as well: openCypher
- Might be an important input to future graph query language standards

openCypher supports two main functions:

- A query language
- An update language

Current specification of openCypher 9:
`https://s3.amazonaws.com/artifacts.opencypher.org/openCypher9.pdf`

Currently not defined in openCypher v9 and depending on implementation:

- Parts of the query language (e.g., comparison and ordering of some values)
- Result formats for sending query results
- Protocol for sending queries and receiving answers

Cypher queries

The heart of Cypher is its query language.

Example .1: The following Cypher query asks for a list of all nodes that are in an `EMPLOYER` relationship:

```
MATCH (person)-[:EMPLOYER]->(company)
RETURN person, company
```

This corresponds to the following SPARQL query:

```
SELECT ?person ?company
WHERE { ?person  :EMPLOYER  ?company }
```

Basic concepts:

- Cypher uses variables, marked by their context of use
- The core of a query is the query condition within `MATCH { ... }`
- Conditions can be simple patterns based on graph edges in a custom syntax
- `RETURN` specifies how results are produced from query matches

Basic Cypher by example

Example .2: Find up to ten people whose daughter is a professor:

```
MATCH
  (parent)-[:HAS_DAUGHTER]->(child {occupation:'Professor'})
RETURN parent
LIMIT 10
```

Basic Cypher by example

Example .2: Find up to ten people whose daughter is a professor:

```
MATCH
  (parent)-[:HAS_DAUGHTER]->(child {occupation:'Professor'})
RETURN parent
LIMIT 10
```

Example .3: Count all relationships in the database:

```
MATCH ()-[relationship]->()
RETURN count(relationship) AS count
```

Basic Cypher by example

Example .2: Find up to ten people whose daughter is a professor:

```
MATCH
  (parent)-[:HAS_DAUGHTER]->(child {occupation:'Professor'})
RETURN parent
LIMIT 10
```

Example .3: Count all relationships in the database:

```
MATCH ()-[relationship]->()
RETURN count(relationship) AS count
```

Example .4: Count all relationship types in the database:

```
MATCH ()-[relationship]->()
RETURN count(DISTINCT type(relationship)) AS count
```

Basic Cypher by example (2)

Example .5: Find the person with most friends:

```
MATCH (person)-[:HAS_FRIEND]->(friend)
RETURN person, count(DISTINCT friend) AS friendCount
ORDER BY friendCount DESC
LIMIT 1
```

Basic Cypher by example (2)

Example .5: Find the person with most friends:

```
MATCH (person)-[:HAS_FRIEND]->(friend)
RETURN person, count(DISTINCT friend) AS friendCount
ORDER BY friendCount DESC
LIMIT 1
```

Example .6: Find pairs of siblings:

```
MATCH
  (parent)-[:HAS_CHILD]->(child1),
  (parent)-[:HAS_CHILD]->(child2)
WHERE id(child1) <> id(child2)
RETURN child1, child2
```

Basic Cypher by example (3): properties and labels

Queries can also access

- Labels (the additional strings used on nodes)
- Properties (of nodes and relationships)

Example .7: Find friends of all people with name Paul Erdős, and return their name and the start date of the friendship:

```
MATCH
(:Human {name: 'Paul Erdős'})-[rel:HAS_FRIEND]->(friend:Human)
RETURN friend.name, rel.startDate
```

Here `Human` is a label, and `name` and `startDate` are property keys.

The shape of a Cypher query

Cypher queries are organised in blocks, called **clauses**:

- Match clause: `MATCH` followed by a pattern; variants of this clause are `OPTIONAL MATCH` and `MANDATORY MATCH`
- Where clause: `WHERE` followed by a filter expression; usually associated with the preceding match clause
- With clause: `WITH` followed by (possibly aliased) expressions and aggregates; ends a subquery
- Return clause: `RETURN` followed by (possibly aliased) expressions and aggregates; occurs once at the end of the query
- Modifier sub-clauses: `ORDER BY`, `LIMIT`, and `SKIP` might follow a With or Return clause
- Union clauses: keyword `UNION` can be used between two complete queries (with `RETURN` for each)

Syntactically, queries are not nested but chained. Many `MATCH-WHERE-WITH` blocks may occur. The order of clauses affects the semantics of queries, but implementations can still evaluate in modified order (as long as the meaning remains the same).

Node patterns

The most basic pattern in Cypher describes a single node.

Definition .8: A node pattern is denoted by a pair of parentheses `()`. It may also contain the following optional components:

- a variable name (a string)
- a list of labels (a list of strings, each prefixed by `:`)
- a set of properties (a comma-separated list of key:value pairs in `{...}`)

Variable names, labels and property keys are quoted with backticks `` ` `` (can be omitted if only alphanumeric characters are used). Property values that are strings are written in straight quotes `'`.

Example .9:

- `()`: an arbitrary node
- `(v { name:'Melitta Bentz', `year of birth`: 1873})`: a node with two properties (and maybe others); matching nodes are bound to variable `v`
- `(:Scientist:Composer)`: a node with two labels

Path patterns

Node patterns are a basic building block of path patterns:

Definition .10: A path pattern is a sequence of one or more node patterns, separated by expressions of the form `-[...]->` (forward) or `<-[...]-` (backward) or `-[...]-` (bidirectional), where the expression in `[...]` is a relationship pattern, with the following optional components:

- a variable name (a string)
- a list of relationship types (`:` followed by a `|`-separated list of strings)
- a range literal (`*`, optionally by a number range $n..m$)
- a set of properties (a comma-separated list of key:value pairs in `{...}`)

Example .11: The following pattern finds nodes `a` and `b` connected by a directed path that consists of between 5 and 10 relationships of types `E` or `F`, where `b` has an incoming relationship `e` with properties that include `score:0.8`:

```
(a)-[:E|F*5..10]->(b)<-[e {score:0.8}]-()
```

Features of path patterns

The various features of path patterns express the following query conditions:

- Sequences of stylised arrows express linear subgraphs (paths with edges in any direction)
- Arrow tips indicate directionality; patterns without any arrow tip match any direction
- The |-separated list of relationship types expresses a disjunction of possible types: the pattern matches if one of the types is found[1]
- The same property-map syntax as in node patterns are used to require the presence of some properties
- The range with * indicates that the specified kind of relationship has to occur multiple times (within a numeric range, where numbers can be omitted to match any finite path)

Note: * always applies to the complete relationship pattern. For example, the disjunction is nested within this iteration.

[1] Recall that relationships in this interpretation of Property Graph can only have one type.

Graph patterns

Path patterns can be combined conjunctively.

Definition .12: A graph pattern is a comma-separated list of path patterns.

Cypher graph patterns are similar to SPARQL basic graph patterns (with property path patterns):

- SPARQL bnodes correspond to Cypher node patterns without variable
- Path patterns are based on similar but slightly distinct features
- Individual path pattern results can be combined with join-like operations to compute overall results

Filter conditions

MATCH clauses can be complemented by **WHERE** clauses that express filters.

Example .13: Find nodes with more than one label:

```
MATCH (n)
WHERE size(labels(n)) > 1
```

As in SPARQL, filters declaratively specify part of the query condition:

- they must be placed after the relevant **MATCH** clause
- but they can be evaluated in any order by a database

Filter conditions

MATCH clauses can be complemented by **WHERE** clauses that express filters.

Example .13: Find nodes with more than one label:

```
MATCH (n)
WHERE size(labels(n)) > 1
```

As in SPARQL, filters declaratively specify part of the query condition:

- they must be placed after the relevant **MATCH** clause
- but they can be evaluated in any order by a database

According to openCypher v9: “If there is a WHERE clause following the match of a shortestPath, relevant predicates will be included in shortestPath.”

Example .14: It is not always clear how to evaluate this efficiently:

```
MATCH p=allShortestPaths((a)-[*]-(b))
WHERE a.someKey + b.someKey < length(p) RETURN p
```

Union

The results of two Cypher queries can be combined on the outermost level.

Example .15: Find parent-child pairs in one of two possible encodings:

```
MATCH (parent) -[:HAS_CHILD]-> (child)
RETURN parent, child
UNION
MATCH (parent) <-[:HAS_PARENT]- (child)
RETURN parent, child
```

“The number and the names of the fields must be identical in all queries combined by using UNION.” (unlike SPARQL)

`UNION` automatically removes duplicates. For keeping them, `UNION ALL` can be used instead.

Optional matches

Similar to SPARQL's `OPTIONAL`, Cypher supports `OPTIONAL MATCH` for clauses that may add additional information, if available.

Example .16: Find parent-child pairs and, optionally, a spouse for the parent:

```
MATCH (parent) -[:HAS_CHILD]-> (child)
OPTIONAL MATCH (parent) -[:SPOUSE]- (spouse)
RETURN parent, child, spouse
```

- If a match cannot be found, then variables will be mapped to null
- Special functions can be used to test for null (e.g., "`WHERE v IS NULL`")
- A graph pattern must match completely, i.e., partial matches will not lead to variable bindings