The Cypher Style Guide

Table of Contents

This is the style guide for the Cypher language, in the context of its standardization through the openCypher project. This document consists of two main sections: Rules and Recommendations.

In Rules, we list syntax guidelines for composing Cypher queries in a conventional, readability-oriented way. The examples provided always transform *valid*, but poorly formatted, queries into a different query in the recommended format, whilst retaining the same semantics.

In Recommendations, we list guidelines that may have an effect on the semantics of queries, such as the way a graph schema is composed through label and relationship types. Bear in mind that these recommendations will not work after-the-fact: if the graph has been constructed with one set of label and relationship types, queries in the associated workload cannot be re-formatted according to these recommendations without also refactoring the data graph.

1. Why is style important?

Consider this dadaist work of art from Nigel Small and Nicole White:

Insane query

```
MATCH (null)-[:merge]->(true)
with null.delete as foreach, `true`.false as null
return 2 + foreach, coalesce(null, 3.1415)
limit 10;
```

Then compare it to this classical piece by Mark Needham:

Sane query

```
MATCH (member:Member {name: 'Mark Needham'})
      -[:HAS_MEMBERSHIP]->()-[:OF_GROUP]->(:Group)-[:HAS_TOPIC]->(topic)
WITH member, topic, count(*) AS score
MATCH (topic)<-[:HAS_TOPIC]-(otherGroup:Group)
WHERE NOT (member)-[:HAS_MEMBERSHIP]->(:Membership)-[:OF_GROUP]->(otherGroup)
RETURN otherGroup.name, collect(topic.name), sum(score) AS score
ORDER BY score DESC
```

The purpose of this document is to help users of the language to share queries with each other with minimal friction, and to construct a consistent and portable usage of the language across many use cases and implementations.

2. Rules

In case two rules are in conflict, and there is no explicit mention of which rule trumps, the rule mentioned last applies.

2.1. Indentation and line breaks

1. Start a new clause on a new line.

   Bad

   ```
   MATCH (n) WHERE n.name CONTAINS 's' RETURN n.name
   ```

   Good

   ```
   MATCH (n)
   WHERE n.name CONTAINS 's'
   RETURN n.name
   ```

   a. Indent `ON MATCH` and `ON CREATE` with two spaces.

      Bad

      ```
      MERGE (n) ON CREATE SET n.prop = 0
      MERGE (a:A)-[:T]-(b:B)
      ON CREATE SET a.name = 'me'
      ON MATCH SET b.name = 'you'
      RETURN a.prop
      ```

Good

```
MERGE (n)
  ON CREATE SET n.prop = 0
MERGE (a:A)-[:T]-(b:B)
  ON CREATE SET a.name = 'me'
  ON MATCH SET b.name = 'you'
RETURN a.prop
```

b. Put `ON CREATE` before `ON MATCH` if both are present.

2. Start a subquery on a new line after the opening brace, indented with two (additional) spaces. Leave the closing brace on its own line.

Bad

```
MATCH (a:A)
WHERE
  EXISTS { MATCH (a)-->(b:B) WHERE b.prop = $param }
RETURN a.foo
```

Also bad

```
MATCH (a:A)
WHERE EXISTS
{MATCH (a)-->(b:B)
WHERE b.prop = $param}
RETURN a.foo
```

Good

```
MATCH (a:A)
WHERE EXISTS {
  MATCH (a)-->(b:B)
  WHERE b.prop = $param
}
RETURN a.foo
```

a. Do not break the line if the simplified subquery form is used.

Bad

```
MATCH (a:A)
WHERE EXISTS {
  (a)-->(b:B)
}
RETURN a.prop
```

Good

```
MATCH (a:A)
WHERE EXISTS { (a)-->(b:B) }
RETURN a.prop
```

2.2. Meta-characters

1. Use single quotes (Unicode character U+0027: ') for literal string values.

   Bad

   ```
   RETURN "Cypher"
   ```

   Good

   ```
   RETURN 'Cypher'
   ```

   a. Disregard this rule for literal strings that contain a single quote character. If the string has both, use the form that creates the fewest escapes. In the case of a tie, prefer single quotes.

      Bad

      ```
      RETURN 'Cypher\'s a nice language', "Mats' quote: \"statement\""
      ```

      Good

      ```
      RETURN "Cypher's a nice language", 'Mats\' quote: "statement"'
      ```

2. Avoid having to use back-ticks to escape characters and keywords.

   Bad

   ```
   MATCH (`odd-ch@racter$`:`Spaced Label` {`&property`: 42})
   RETURN labels(`odd-ch@racter$`)
   ```

Good

```
MATCH (node:NonSpacedLabel {property: 42})
RETURN labels(node)
```

3. Do not use a semicolon at the end of the statement.

Bad

```
RETURN 1;
```

Good

```
RETURN 1
```

2.3. Casing

1. Write keywords in upper case.

Bad

```
match (p:Person)
where p.name starts with 'Ma'
return p.name
```

Good

```
MATCH (p:Person)
WHERE p.name STARTS WITH 'Ma'
RETURN p.name
```

2. Write the value `null` in lower case.

Bad

```
WITH NULL AS n1, Null AS n2
RETURN n1 IS NULL AND n2 IS NOT NULL
```

Good

```
WITH null AS n1, null as n2
RETURN n1 IS NULL AND n2 IS NOT NULL
```

3. Write boolean literals in lower case.

Bad

```
WITH TRUE AS b1, False AS b2
RETURN b1 AND b2
```

Good

```
WITH true AS b1, false AS b2
RETURN b1 AND b2
```

4. Use camel case, starting with a lower case character, for:
   a. functions
   b. properties
   c. variables
   d. parameters

   Bad

   ```
   CREATE (N {Prop: 0})
   WITH RAND() AS Rand, $pArAm AS MAP
   RETURN Rand, MAP.property_key, Count(N)
   ```

   Good

   ```
   CREATE (n {prop: 0})
   WITH rand() AS rand, $param AS map
   RETURN rand, map.propertyKey, count(n)
   ```

2.4. Patterns

1. When patterns wrap lines, break after arrows, not before.

   Bad

   ```
   MATCH (:Person)-->(vehicle:Car)-->(:Company)
         <--(:Country)
   RETURN count(vehicle)
   ```

   Good

   ```
   MATCH (:Person)-->(vehicle:Car)-->(:Company)<--
         (:Country)
   RETURN count(vehicle)
   ```

2. Use anonymous nodes and relationships when the variable would not be used.

   Bad

   ```
   CREATE (a:End {prop: 42}),
          (b:End {prop: 3}),
          (c:Begin {prop: id(a)})
   ```

   Good

   ```
   CREATE (a:End {prop: 42}),
          (:End {prop: 3}),
          (:Begin {prop: id(a)})
   ```

3. Chain patterns together to avoid repeating variables.

   Bad

   ```
   MATCH (:Person)-->(vehicle:Car), (vehicle:Car)-->(:Company)
   RETURN count(vehicle)
   ```

   Good

   ```
   MATCH (:Person)-->(vehicle:Car)-->(:Company)
   RETURN count(vehicle)
   ```

4. Put named nodes before anonymous nodes.

   Bad

   ```
   MATCH ()-->(vehicle:Car)-->(manufacturer:Company)
   WHERE manufacturer.foundedYear < 2000
   RETURN vehicle.mileage
   ```

   Good

   ```
   MATCH (manufacturer:Company)<--(vehicle:Car)<--()
   WHERE manufacturer.foundedYear < 2000
   RETURN vehicle.mileage
   ```

5. Keep anchor nodes at the beginning of the `MATCH` clause.

Bad

```
MATCH (:Person)-->(vehicle:Car)-->(manufacturer:Company)
WHERE manufacturer.foundedYear < 2000
RETURN vehicle.mileage
```

Good

```
MATCH (manufacturer:Company)<--(vehicle:Car)<--(:Person)
WHERE manufacturer.foundedYear < 2000
RETURN vehicle.mileage
```

6. Prefer outgoing (left to right) pattern relationships to incoming pattern relationships.

Bad

```
MATCH (:Country)-->(:Company)<--(vehicle:Car)<--(:Person)
RETURN vehicle.mileage
```

Good

```
MATCH (:Person)-->(vehicle:Car)-->(:Company)<--(:Country)
RETURN vehicle.mileage
```

2.5. Spacing

1. For literal maps:
   a. No space between the opening brace and the first key
   b. No space between key and colon
   c. One space between colon and value
   d. No space between value and comma
   e. One space between comma and next key
   f. No space between the last value and the closing brace

Bad

```
WITH { key1 :'value' ,key2  :  42 } AS map
RETURN map
```

Good

```
WITH {key1: 'value', key2: 42} AS map
RETURN map
```

2. No padding space for parameters.
   a. This rule mentions deprecated syntax. See Parameter Syntax.

Bad

```
RETURN { param }
```

Good

```
RETURN {param}
```

3. One space between label/type predicates and property predicates in patterns.

Bad

```
MATCH (p:Person{property: -1})-[:KNOWS   {since: 2016}]->()
RETURN p.name
```

Good

```
MATCH (p:Person {property: -1})-[:KNOWS {since: 2016}]->()
RETURN p.name
```

4. No space in patterns.

Bad

```
MATCH (:Person) --> (:Vehicle)
RETURN count(*)
```

Good

```
MATCH (:Person)-->(:Vehicle)
RETURN count(*)
```

5. Use a wrapping space around operators.

Bad

```
MATCH p=(s)-->(e)
WHERE s.name<>e.name
RETURN length(p)
```

Good

```
MATCH p = (s)-->(e)
WHERE s.name <> e.name
RETURN length(p)
```

6. No space in label predicates.

Bad

```
MATCH (person    : Person  :  Owner  )
RETURN person.name
```

Good

```
MATCH (person:Person:Owner)
RETURN person.name
```

7. Use a space after each comma in lists and enumerations.

Bad

```
MATCH (),()
WITH ['a','b',3.14] AS list
RETURN list,2,3,4
```

Good

```
MATCH (), ()
WITH ['a', 'b', 3.14] AS list
RETURN list, 2, 3, 4
```

8. No padding space within function call parentheses.

Bad

```
RETURN split( 'original', 'i' )
```

Good

```
RETURN split('original', 'i')
```

9. Use padding space within simple subquery expressions.

Bad

```
MATCH (a:A)
WHERE EXISTS {(a)-->(b:B)}
RETURN a.prop
```

Good

```
MATCH (a:A)
WHERE EXISTS { (a)-->(b:B) }
RETURN a.prop
```

3. Recommendations

- When using Cypher language constructs in prose, use a monospaced font and follow the styling rules.
  - When referring to labels and relationship types, the colon should be included as follows: `:Label`, `:REL_TYPE`.
  - When referring to functions, use lower camel case and parentheses should be used as follows: `shortestPath()`. Arguments should normally not be included.
- If you are storing Cypher statements in a separate file, use the file extension `.cypher`.

3.1. Graph modelling

1. Prefer single nouns for labels.

Bad

```
MATCH (e:IsEmployed)
RETURN e.name
```

Good

```
MATCH (e:Employee)
RETURN e.name
```

2. Write labels in camel case, starting with an upper case character.

Bad

```
MATCH (e:editor_in_chief)-->(:EMPLOYEE)
RETURN e.name
```

Good

```
MATCH (e:EditorInChief)-->(:Employee)
RETURN e.name
```

3. Write relationship types in upper case, using an underscore (_) to separate words.

Bad

```
MATCH (:Person)-[own:ownsVehicle]->(:Car)
RETURN own.since
```

Good

```
MATCH (:Person)-[own:OWNS_VEHICLE]->(:Car)
RETURN own.since
```