User study - Property Graph Transformations

* Indique une question obligatoire

Researchers

Angela Bonifati (angela.bonifati@univ-lyon1.fr)
Filip Mulak (fmurlak@mimuw.edu.pl)
Yann Ramusat (yann.ramusat@liris.cnrs.fr) **- Corresponding**

Purpose of the study

You are being invited to participate in a research study from the Lyon1 University. Participation is limited to adults (18 years of age and older). The purpose of this test is to assess the usability of our framework for specifying transformations of property graphs. This study will take you approximately 30 minutes to complete. The data will be used to assess the efficiency and the quality of our approach and compare it to existing openCypher solutions.

Benefits and risks of participation

The risks associated with participation in this study are no greater than those ordinarily encountered in daily life or during other online activities. There may be no direct benefit to you. However, the research result that comes from your data can contribute to design better courses and more understandable query languages.

Procedures for withdrawal from the study

Your participation in this study is entirely voluntary and you can withdraw at any time.

Data collection and processing

We will collect only the data you submit. For the purpose of this study your identifiers will be deleted after the research is done.

Data usage and privacy

The analysis results and the data collection methodology may be published in research papers or technical reports. We will aim for open access publication. During the active phase of research, researchers from Lyon1 will oversee the access rights to data (and other outputs), as well as any requests for access from external parties. Sharing of data with others will only be done in such a manner - and never for the raw data. The subjects will not be identified. We will host data on password-protected storages.

Research participant rights

If you have concerns or questions about this research study, or if you believe that you have been harmed due to participation in this study, please contact the researchers, listed at the beginning of this informed consent form.

By continuing to fill in the following form you consent to participate to the user study.

1. How would you rate your knowledge about databases? *

   Une seule réponse possible.

   - 1 - Novice
   - 2
   - 3
   - 4
   - 5 - Expert

2. How would you rate your knowledge about openCypher? *

   Une seule réponse possible.

   - 1 - Novice
   - 2
   - 3
   - 4
   - 5 - Expert

3. How would you rate your knowledge about the MERGE clause of openCypher? *

   Une seule réponse possible.

   - 1 - Novice
   - 2
   - 3
   - 4
   - 5 - Expert

4. How would you rate your knowledge about property graph transformations? *

Une seule réponse possible.

- ⬭ 1 - Novice
- ⬭ 2
- ⬭ 3
- ⬭ 4
- ⬭ 5 - Expert

Study procedure and structure

The study begins with a few questions regarding your understanding of some openCypher scripts in constructing property graph instances.

Then it continues with a tutorial presenting our framework. This framework allows users to specify *transformations* of property graphs (i.e. mappings between property graph instances). You will then be asked a few questions regarding your understanding of some transformations expressed in our framework.

Second, you will need to modify some provided scripts and transformations in order to repair them to meet a specific transformation requirement.

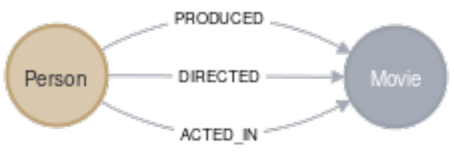
Finally, you will be asked some questions about your feelings and thoughts regarding using both approaches.

Keep in mind that you won't be evaluated based on your answers.

Understandability of openCypher scripts

Our data describes a scenario where people can either be actors, producers or directors of movies. A node represents a *Person* or a *Movie* and the relationships of type *PRODUCED*, *DIRECTED* or *ACTED_IN* give the role(s) of a person in a movie.

Schema:



The objective is to write in openCypher a script to produce a new property graph where the *Person* nodes are split into (possibly overlapping) sets of nodes with labels *Actors*, *Directors* and *Producers*.

In the following questions, you can select Other in case you don't want to give a random answer, either you have no idea or you think the specification isn't clear enough. You must provide a brief explanation. The correct answer is always Yes or No.

Here are some questions concerning the output of the following queries.

To reshape the data, suppose the user enters these two queries, one after the other:

```
MATCH (n:Person)-[:DIRECTED]->(:Movie)
CREATE (:Director {
   name: n.name,
   born: n.born
})
```

and

```
MATCH (n:Person)-[:ACTED_IN]->(:Movie)
CREATE (:Actor {
   name: n.name,
   born: n.born
})
```

Note: link to Cypher Manual -- CREATE clause

5. Does this query create as many *Director* nodes as there are *Person* nodes that have an outgoing relationship of type *DIRECTED* to a *Movie* node? *

Une seule réponse possible.

- Yes
- No
- Autre : ______________________

6. Does this query can create nodes with both *Director* and *Actor* labels? *

Une seule réponse possible.

- Yes
- No
- Autre : ______________________

Now suppose the user executes the following script after having executed the two previous ones:

```
MATCH (n:Person)-[:DIRECTED]->(m:Movie)
MERGE (:Director {
  name: n.name,
  born: n.born
})-[:DIRECTOR_OF]-(:Film {
  title: m.title
})
```

Note: link to Cypher Manual -- MERGE clause

7. Executed after the previous two queries, does this one will create new *Director* nodes? *

Une seule réponse possible.

- Yes
- No
- Autre : ______________________

8. Will two *Persons* having co-directed the same *Movie* be connected to a same *Film* node? *

Une seule réponse possible.

- Yes
- No
- Autre : ______________________

Understandability of Property Graph Transformations

Tutorial

In the following, we provide a brief tutorial covering the essential aspects of our property graph transformation framework. Its implementation takes the form of a new clause for openCypher called GENERATE.

The following example illustrates a *transformation rule* (an openCypher query that uses the GENERATE clause):

```
MATCH (n:FirstName)-[r]->(m:LastName)
GENERATE
(x = (r):FullName {
   value = n.value + m.value
})
```

The GENERATE clause has a dedicated part (here *x = (r)*) to specify, using a list of arguments, the identity of an element to be created.
In this specific case, for each binding of *r* produced by the MATCH clause, a new element with label *FullName* will be created.

E.g.: If the source property graph contains the following three nodes and two relations

```
(id1:FirstName {
   value: "a"
})-[id2]->(id3:LastName {
   value: "b"
})<-[id4]-(id5:FirstName {
   value: "c"
})
```

The output of the previous transformation rule contains two nodes:

```
(new_id1:FullName {
   value: "ab"
})
```

and

```
(new_id2:FullName {
   value: "cb"
})
```

Explanation: *new_id1* has been generated for the id list (id2) and *new_id2* has been generated for the id list (id4) because *r* binds to id2 and id4.
Note: the + denotes string concatenation.

Using list of identifiers

The identity of a new element is a (possibly empty) list of:

- identifier (e.g. *r*)
- a constant data value (e.g. a string *"Robert"*)
- an access key (e.g. *n.value*)
- a label (e.g. *FullName*)

An added advantage of using this framework is that several rules (i.e. GENERATE clauses) share their scope:

```
MATCH (n:FirstName)
GENERATE
(x = (n.value):Name {
   value = n.value,
   isFirstName = "yes"
})
```

and

```
MATCH (n:LastName)
GENERATE
(x = (n.value):Name {
   value = n.value,
   isLastName = "yes"
})
```

Assuming that it exists in the data a first name (i.e. *n.value*) which is also a last name, an element with label *Name* will be created with both its properties *isFirstName* and *isLastName* set to *"yes"*.

E.g.: If the source property graph contains the following three nodes:

```
(id1:LastName {
   value: "1"
})
(id2:FirstName {
   value: "1"
})
```
and
```
 (id3:FirstName {
   value: "2"
})
```

The output of the two previous rules contains two nodes:

```
(new_id1:Name {
   value: "1",
   isLastName: "yes",
   isFirstName: "yes"
})
```
and
```
(new_id1:Name {
   value: "2",
   isFirstName = "yes"
})
```

Explanation: *new_id1* has been generated for the id list ("1") and *new_id2* has been generated for the id list ("2") because *n.value* takes the values "1" for the bindings of *n* which are *id1* and *id2* and takes the value "2" for the binding of *n* which is *id3*.

Creating relationships

One can also specify relationships in this framework such as with the following rule:

```
MATCH (n:FirstName)-[r]->(m:LastName)
GENERATE
(x = (n.value):Name {
   value = n.value
})-[():FullName {
   value = n.value + m.value
}]->(y = (m.value):Name {
  value = m.value
})
```

In this situation, for each relationship between a *FirstName* and a *LastName*, at most one relationship will be created between their corresponding *Name* nodes.

E.g.: If the source property graph contains the following three nodes and two relations

```
(id1:FirstName {
   value: "a"
})-[id2]->(id3:LastName {
   value: "b"
})<-[id4]-(id5:FirstName {
   value: "a"
})
```

The output of the previous transformation rule contains two nodes and one relationship:

```
(new_id1:Name {
   value: "a"
})-[new_id2:FullName {
   value: "ab"
]->(new_id3:Name {
   value: "b"
})
```

Explanation: *new_id1* has been generated for the id list ("a") and *new_id3* has been generated for the id list ("b") because *n.value* takes the values "a" for the bindings of *n* which are *id1* and *id5* and takes the value "b" for the binding of m which is *id3*. *new_id2* has been generated for the id list (*new_id1*, *new_id3)* which consists of its two endpoints in the output property graph.

Creating multiple relationships between the same endpoints

If one wants to have many relationships between two endpoints in the example above, one could do:

```
MATCH (n:FirstName)-[r]->(m:LastName)
GENERATE
(x = (n.value):Name)-[(r):FullName {
   value = n.value + m.value
}]->(y = (m.value):Name)
```

E.g.: If the source property graph contains the following three nodes and two relations

```
(id1:FirstName {
   value: "a"
})-[id2]->(id3:LastName {
   value: "b"
})<-[id4]-(id5:FirstName {
   value: "a"
})
```

The output of the previous transformation rule contains two nodes

```
(new_id1:Name {
   value: "a"
})
```

and

```
(new_id3:Name {
   value: "b"
})
```

and there are two relationships between them

```
(new_id1)-[new_id2:FullName {
   value: "ab"
]->(new_id3)
```

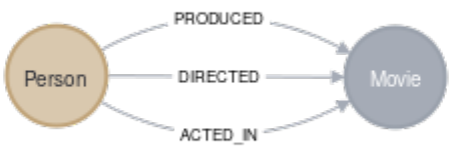
and

```
(new_id1)-[new_id4:FullName {
   value: "ab"
]->(new_id3)
```

Explanation: *new_id1* has been generated for the id list ("a") and *new_id3* has been generated for the id list ("b") because *n.value* takes the values "a" for the bindings of *n* which are *id1* and *id5* and takes the value "b" for the binding of m which is *id3*.
new_id2 has been generated for the id list (*new_id1*, *id2, new_id3)* which consists of its two endpoints in the output property graph and the binding of *r* to *id2*.
new_id4 has been generated for the id list (*new_id1*, *id4, new_id3)* which consists of its two endpoints in the output property graph and the binding of *r* to *id4*.

We remind you that our data describes a scenario where people can either be actors, producers or directors of movies. A node represents a *Person* or a *Movie* and the relationships of type *PRODUCED*, *DIRECTED* or *ACTED_IN* give the role(s) of a person in a movie.

Schema:



The objective is still to produce a new property graph where the *Person* nodes are split into (possibly overlapping) sets of nodes with labels *Actors*, *Directors* and *Producers*.

In the following questions, you can select Other in case you don't want to give a random answer, either you have no idea or you think the specification isn't clear enough. You must provide a brief explanation. The correct answer is always Yes or No.

Suppose the user writes and executes the transformation consisting of the following two rules:

```
MATCH (n:Person)-[:ACTED_IN]->(:Movie)
GENERATE
((n):Actor {
   name = n.name,
   born = n.born
})
```

and

```
MATCH (n:Person)-[:DIRECTED]->(:Movie)
GENERATE
((n):Director {
   name = n.name,
   born = n.born
})
```

9. Does this transformation generate as many *Director* nodes as there are *Person* nodes that have an outgoing relationship of type *DIRECTED* to a *Movie* node? *

Une seule réponse possible.

- Yes
- No
- Autre : ______________________

10. Does this transformation can generate nodes with both a *Director* and an *Actor* labels? *

Une seule réponse possible.

- Yes
- No
- Autre : ______________________

Now suppose the user adds a new rule to this transformation:

```
MATCH (n:Person)-[:DIRECTED]->(m:Movie)
GENERATE
((n):Director {
  name = n.name,
  born = n.born
})-[():DIRECTOR_OF]->((m):Film {
  title = m.title
})
```

11. Will more *Director* nodes be created? *

Une seule réponse possible.

- Yes
- No
- Autre : ______________________

12. Will two persons having co-directed the same movie be connected to the same *Film* node? *

Une seule réponse possible.

- Yes
- No
- Autre : ____________________

Conflicting property values with openCypher scripts and transformation rules

Let us now consider the following openCypher script and the following transformation rule which are equivalent:

```
MATCH (n:Person)-[:DIRECTED]->(m:Movie)<-[:DIRECTED]-(o:Person)
MERGE (x:Director {
   name: n.name,
   born: n.born
})
MERGE (y:Director {
   name: o.name,
   born: o.born
})
MERGE (x)-[d:COLLEAGUE]->(y)
SET d.movie = m.title
```

and

```
MATCH (n:Person)-[:DIRECTED]->(m:Movie)<-[:DIRECTED]-(o:Person)
GENERATE
(x = (n.name, n.born):Director {
   name = n.name,
   born = n.born
})-[():COLLEAGUE {
   movie = m.title
}]->(y = (o.name, o.born):Director {
   name = o.name,
   born = o.born
})
```

Unfortunately, the value of the *movie* attribute of some *COLLEAGUE* relationships may not be well-defined. This is for instance the case when two directors have co-directed several movies because the *movie* attribute can only store one title.

First approach

To solve this, we now want to have as many relationships between two *Directors* as there are possible values for the *movie* attribute (i.e. distinct titles for *Movies* that have been co-directed by the two *Directors*).

13. How would you modify the openCypher script to account for this? *
Please provide the modified openCypher script.

14. How would you modify the transformation rule to account for this? *
Please provide the modified transformation rule.

Second approach

We now want to solve this in a different way. We want one relationship for each *Movie* that has been co-directed by the two *Directors*.

15. How would you modify the openCypher script to account for this? *
Please provide the modified openCypher script.

16. How would you modify the transformation rule to account for this? *
Please provide the modified transformation rule.

Final thoughts

We now ask you some questions about your feelings using both approaches.

17. From a scale of 1 to 5, which one of the two methods you find easier to understand? (When a transformation is given, inferring the produced output.) *

Une seule réponse possible.

- 1 - openCypher scripts are easier to understand
- 2
- 3
- 4
- 5 - The transformations rules and the GENERATE clause are easier to understand

18. Why?

19. From a scale of 1 to 5, which one of the two methods you find more intuitive? *
(Better for describing the desired output.)

Une seule réponse possible.

- 1 - openCypher scripts better describe the intended output
- 2
- 3
- 4
- 5 - The transformations rules and the GENERATE clause better describe the intended output

20. Why?

21. From a scale of 1 to 5, which one of the two methods you find more flexible? *
(Easier to adapt to a new specification.)

Une seule réponse possible.

- 1 - openCypher scripts are easier to adapt
- 2
- 3
- 4
- 5 - The transformations rules and the GENERATE clause are easier to adapt

22. Why?