Big Graph Processing Systems

Part II: Property Graphs

▶ Chapter 3: Schema Discovery and Property Graph Transformations

Christopher Spinrath

CNRS – LIRIS – Lyon 1 Université

DISS Master 2025

This presentation is an adaption of slides from Angela Bonifati

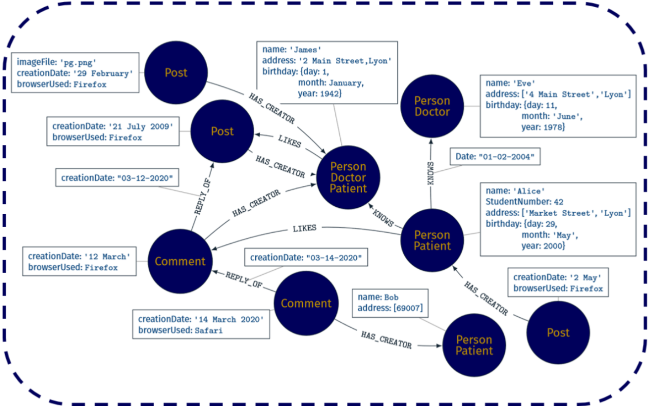
Schema Discovery

From Big Data to Machine Learning

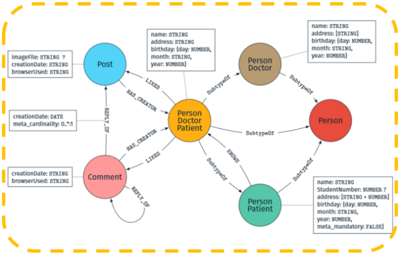
the-matrix

Schema Discovery

Input PG



Output PG Schema



Schema Discovery for Property Graphs

Existing schema discovery/inference mechanisms are basic

KellouMenouer2022

- no hierarchies
- no complex types

Schema Discovery for Property Graphs

Existing schema discovery/inference mechanisms are basic

KellouMenouer2022

- no hierarchies
- no complex types

MRSchema: Schema inference using MapReduce on Spark

Lbath2021

Code Base: `https://gitlab.com/Hgit/pgsinference`

- considers either node labels or node properties $\rightarrow$ trade-off
- property co-occurrence information loss (label-oriented approach) vs. extraneous type inference (property-oriented approach)

Schema Discovery for Property Graphs

Existing schema discovery/inference mechanisms are basic

KellouMenouer2022

- no hierarchies
- no complex types

MRSchema: Schema inference using MapReduce on Spark

Lbath2021

Code Base: `https://gitlab.com/Hgit/pgsinference`

- considers either node labels or node properties $\rightarrow$ trade-off
- property co-occurrence information loss (label-oriented approach) vs. extraneous type inference (property-oriented approach)

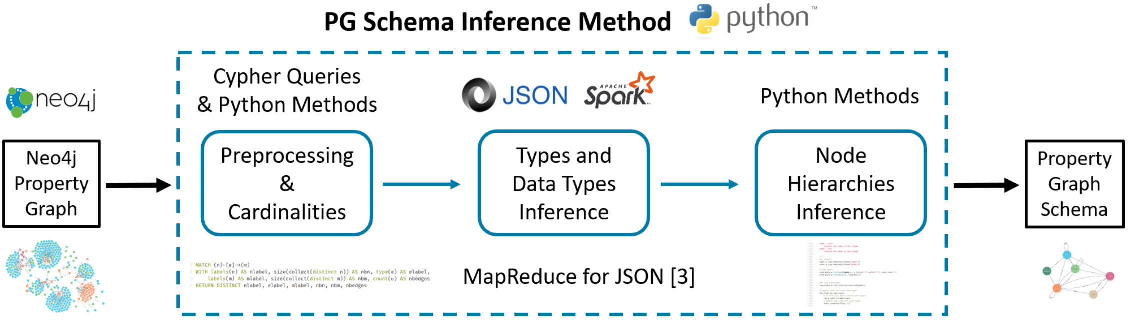
Schema inference using hierarchical clustering

Bonifati2022

Code Base: `https://github.com/PI-Clustering/code`

- Can handle labels and properties at the same time

Overview of the MRSchema Method



Two Variants

- Label-oriented: label sets characterize types
- Property-oriented: labels are properties, property key sets characterize types

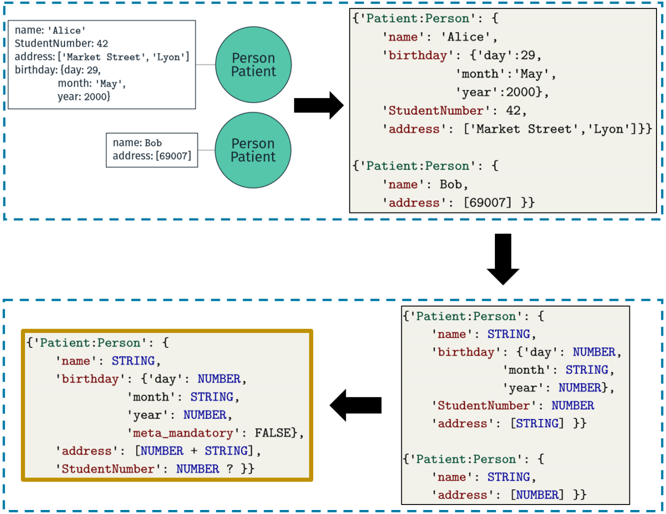
MRSchema – Step 1 and Step 2

Step 1: Preprocessing & Cardinalities

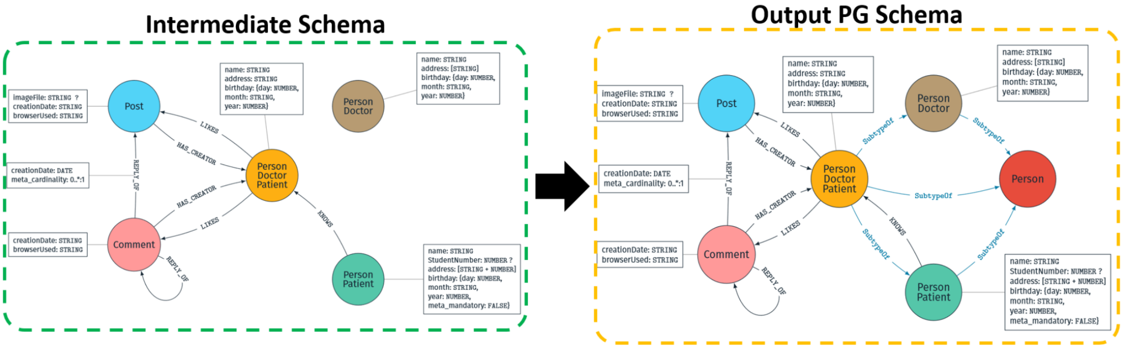
- Convert input PG to proper format
- Infer edge cardinality constraints

Step 2: Types & Data Types Inference (MapReduce)

- Label sets characterize types

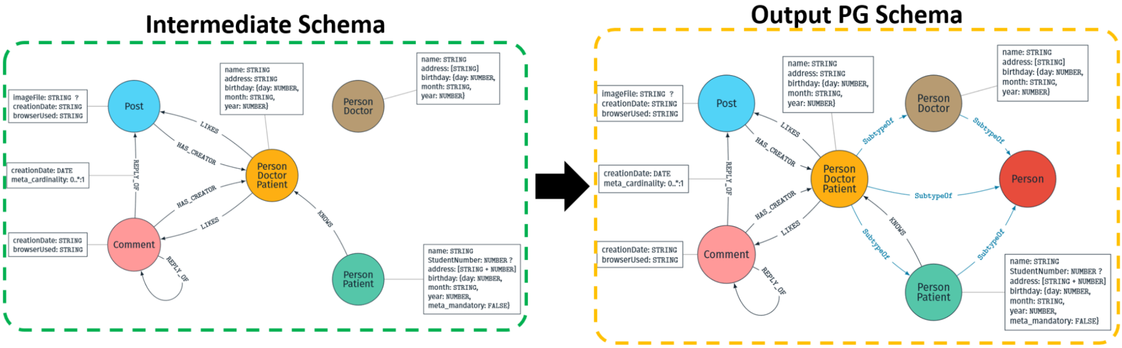


Step 3: Node Hierarchies Inference (Label-oriented variant)



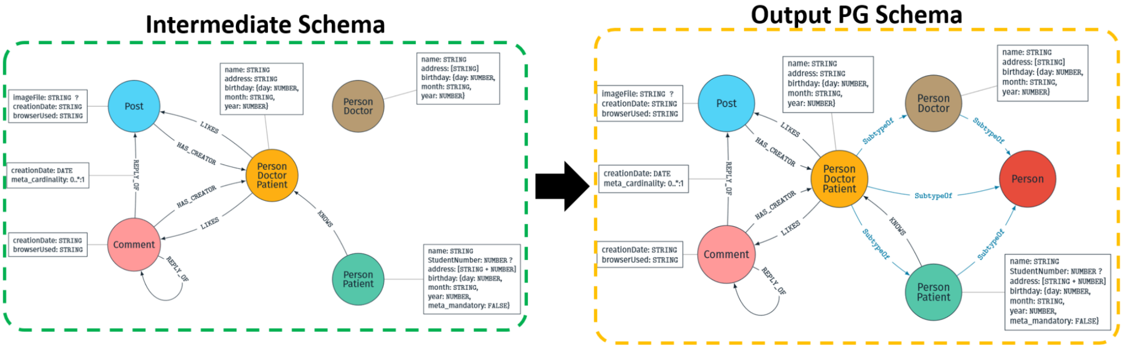
Step 3: Node Hierarchies Inference (Label-oriented variant)

- Supertype inference: Pairwise intersection of label sets

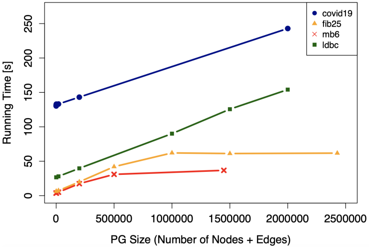


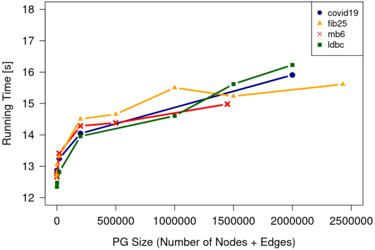
Step 3: Node Hierarchies Inference (Label-oriented variant)

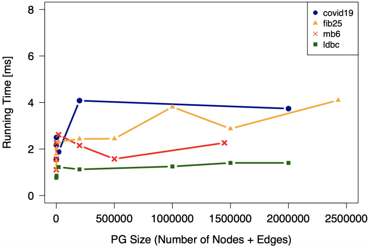
- Supertype inference: Pairwise intersection of label sets
- Subtype inference: Node type with label set A is a subtype of node type with label set B if $B \subsetneq A$



MRSchema – Time Performances (per step)







Cypher Queries
& Python Methods

JSON Spark

Python Methods

Preprocessing & Cardinalities → Types and Data Types Inference → Node Hierarchies Inference

MRSchema – Property-Oriented Variant

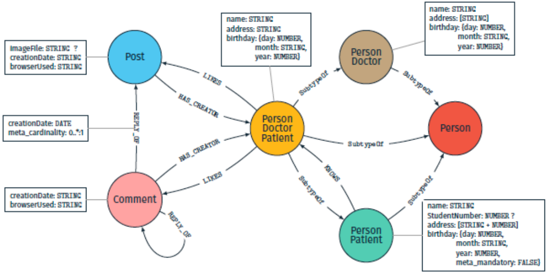
Property-Oriented Variant

Labels are properties, property key sets characterize node types

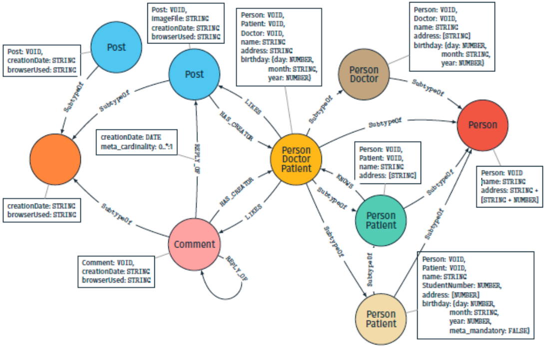
- **Step 1:** Unlabelled nodes are also matched
- **Step 2:** Identification of property co-occurrence information but not optional properties
- **Step 3:** Property key sets are used for subtypes and supertypes inference

MRSchema – Label-Oriented vs. Property-Oriented Variant

Schema derived with
the label-oriented variant



Schema derived with
the property-oriented variant



A New Clustering-based Method: The DiscoPG System

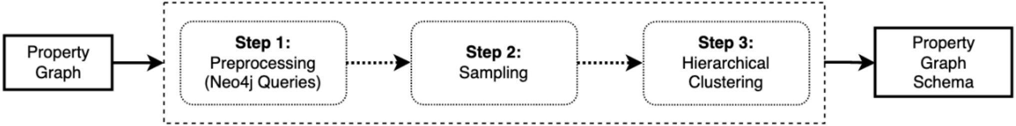
- Need of combining labels and properties for type inference with improved precision and recall
- Static Case: discover the schema of a static graph dataset G
  - GMM-S: novel hierarchical clustering algorithm
  - Based on fitting a Gaussian Mixture Model (GMM)
  - Accounts for both node label & property information
- Dynamic Case: update the schema of G upon modifications
  - I-GMM-D: incremental approach; reuses GMM-S clustering
  - GMM-D: recomputation approach; memorization-based

Bonifati2022

Bonifati2022a

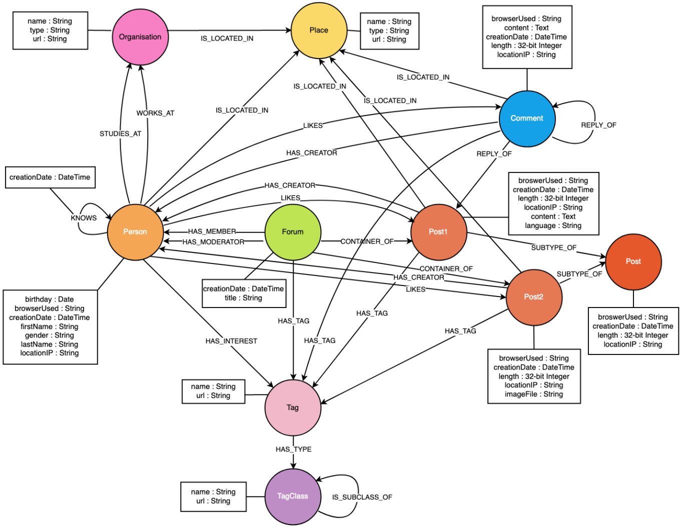
A GMM Schema Pipeline

- Gaussian Mixture Model (GMM*) to discover hierarchical node types
- For every node label, run GMM algorithm to fit a mixture of normal distributions and use the resulting model for clustering
- Re-iterate procedure in each sub-cluster



*Dempster1977

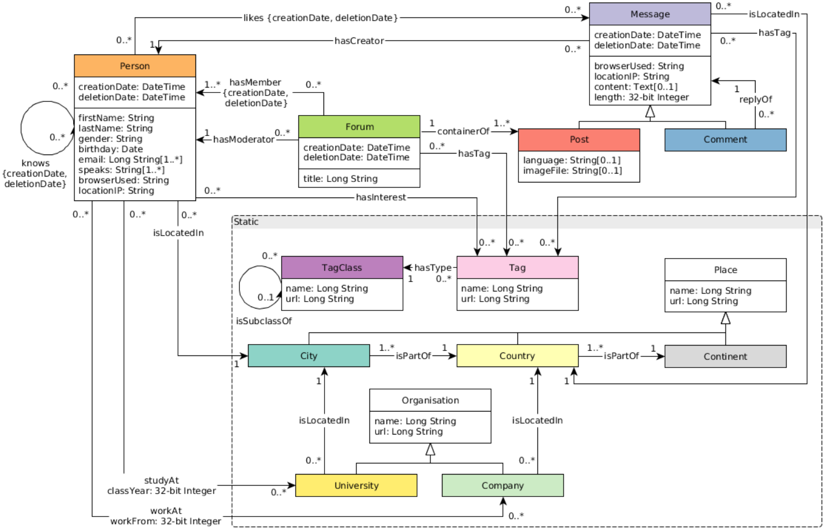
Inferred LDBC Schema



Note

Base type Post has two subtypes Post1 and Post2

LDBC Schema



Message
creationDate: DateTime
deletionDate: DateTime
browserUsed: String
locationIP: String
content: Text[0..1]
length: 32-bit Integer
likes {creationDate, deletionDate}
hasCreator
isLocatedIn
hasTag
replyOf
Person
creationDate: DateTime
deletionDate: DateTime
firstName: String
lastName: String
gender: String
birthday: Date
email: Long String[1..*]
speaks: String[1..*]
browserUsed: String
locationIP: String
knows
{creationDate,
deletionDate}
hasMember
{creationDate,
deletionDate}
hasModerator
Forum
creationDate: DateTime
deletionDate: DateTime
title: Long String
containerOf
Post
language: String[0..1]
imageFile: String[0..1]
Comment
hasInterest
Static
TagClass
name: Long String
url: Long String
hasType
Tag
name: Long String
url: Long String
isSubclassOf
Place
name: Long String
url: Long String
City
isPartOf
Country
Continent
Organisation
name: Long String
url: Long String
University
Company
studyAt
classYear: 32-bit Integer
workAt
workFrom: 32-bit Integer

Schema Quality wrt. Baseline (MRSchema)

MRSchema
property-oriented variant

Dataset	Node Types	Edge Types	Subtype Edges	Hierarchy Depth
LDBC	17	72	51	5
Mb6	68	795	786	9
Fib25	47	427	418	8

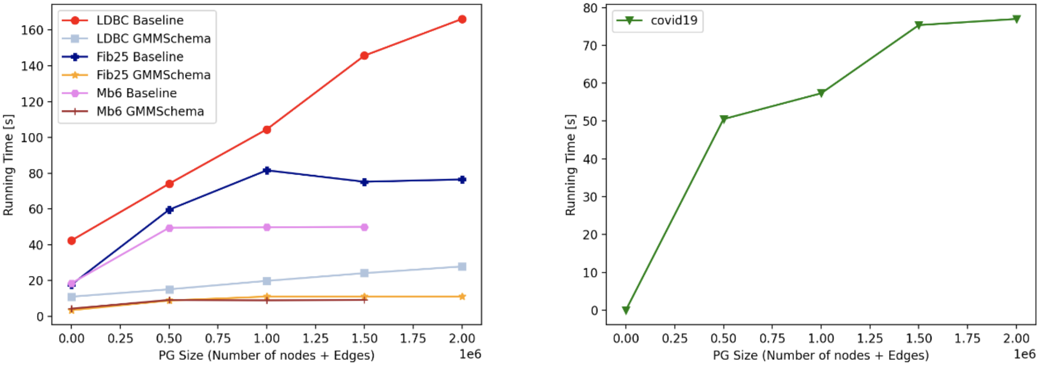
MRSchema
label-oriented variant

Dataset	Node Types	Edge Types	Subtype Edges	Hierarchy Depth
LDBC	7	21	0	0
Mb6	5	10	1	1
Fib25	5	10	1	1

GMMSchema

Dataset	Node Types	Edge Types	Subtype Edges	Hierarchy Depth
LDBC	17	36	9	2
Mb6	19	27	14	4
Fib25	26	106	21	6

GMM Schema Discovery Runtimes wrt. Baseline



(a) LDBC, Fib25, Mb6

(b) Covid19

Property Graph Transformations

A Declarative Transformation Framework

source-code

(DBLP:journals/pvldb/BonifatiMR24)

Declarative Property Graph Transformations Are Essential

The property graph data model is flexible and agile

- Schema-last approach; as opposed to the schema-first approach in SQL
- The representation of the data depends on the evolving use-cases

Declarative Property Graph Transformations Are Essential

The property graph data model is flexible and agile

- Schema-last approach; as opposed to the schema-first approach in SQL
- The representation of the data depends on the evolving use-cases

Requirements for Property Graph Transformations

1. Nodes may be transformed into edges (and vice-versa)

Declarative Property Graph Transformations Are Essential

The property graph data model is flexible and agile

- Schema-last approach; as opposed to the schema-first approach in SQL
- The representation of the data depends on the evolving use-cases

Requirements for Property Graph Transformations

1. Nodes may be transformed into edges (and vice-versa)
2. A new piece of data may represent a complex pattern over the source data
    - Defining an appropriate notion of identity is crucial

Declarative Property Graph Transformations Are Essential

The property graph data model is flexible and agile

- Schema-last approach; as opposed to the schema-first approach in SQL
- The representation of the data depends on the evolving use-cases

Requirements for Property Graph Transformations

1. Nodes may be transformed into edges (and vice-versa)
2. A new piece of data may represent a complex pattern over the source data
   - Defining an appropriate notion of identity is crucial
3. Primitives for handling data contents
   - Data values may contribute to the identity of new elements
   - Similarly, labels may contribute to the identity of new elements

Declarative Property Graph Transformations Are Essential

The property graph data model is flexible and agile

- Schema-last approach; as opposed to the schema-first approach in SQL
- The representation of the data depends on the evolving use-cases

Requirements for Property Graph Transformations

1. Nodes may be transformed into edges (and vice-versa)
2. A new piece of data may represent a complex pattern over the source data
   - Defining an appropriate notion of identity is crucial
3. Primitives for handling data contents
   - Data values may contribute to the identity of new elements
   - Similarly, labels may contribute to the identity of new elements
4. New elements may aggregate the content of past ones

Current State

- Typical graph query languages return tuples (rows of a table)
- Hence, they cannot be composed/chained together
- A transformation should rather return a (sub)graph structure

Current State

- Typical graph query languages return tuples (rows of a table)
- Hence, they cannot be composed/chained together
- A transformation should rather return a (sub)graph structure

Existing Solutions and Approaches

- In practical graph database systems such as Neo4j
  - Handcrafted openCypher scripts
  - APOC (Awesome Procedures on Cypher)

Current State

- Typical graph query languages return tuples (rows of a table)
- Hence, they cannot be composed/chained together
- A transformation should rather return a (sub)graph structure

Existing Solutions and Approaches

- In practical graph database systems such as Neo4j
  - Handcrafted openCypher scripts
  - APOC (Awesome Procedures on Cypher)
- In the research literature
  - Data exchange for graph databases
    10.1145/2448496.2448520
    10.1145/3034786.3056113
  - Graph databases transformations
    10.1145/3584372.3588654

Current State

- Typical graph query languages return tuples (rows of a table)
- Hence, they cannot be composed/chained together
- A transformation should rather return a (sub)graph structure

Existing Solutions and Approaches

- In practical graph database systems such as Neo4j
  - Handcrafted openCypher scripts
  - APOC (Awesome Procedures on Cypher)
- In the research literature
  - Data exchange for graph databases
    10.1145/2448496.2448520
    10.1145/3034786.3056113
  - Graph databases transformations
    10.1145/3584372.3588654

None are declarative!

Property Graph Transformations

Declarative Specifications

… have been recognized as pivotal for solving data programmability problems

bernstein_model_2007

Property Graph Transformations

Declarative Specifications

... have been recognized as pivotal for solving data programmability problems

bernstein_model_2007

New Framework for Property Graph Transformations

DBLP:journals/pvldb/BonifatiMR24

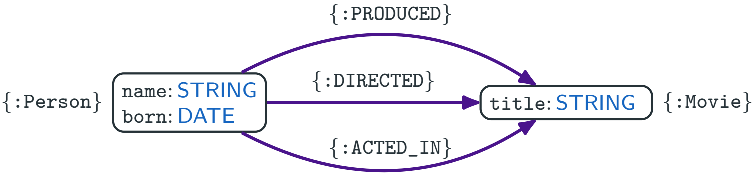
Demo: **DBLP:journals/pvldb/BonifatiRMFE24**

Code Base: `https://github.com/yannramusat/DTGraph/`

- Declarative; rule-based
- Intuitive and expressive
- Efficiently implementable in practical graph database systems
- openCypher extension and theoretical foundations (GPC)

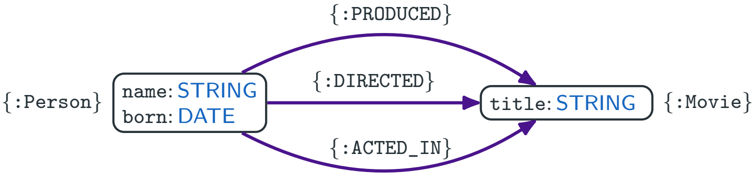
DTGraph – An Example of the GENERATE clause

Input Schema



DTGraph – An Example of the GENERATE clause

Input Schema



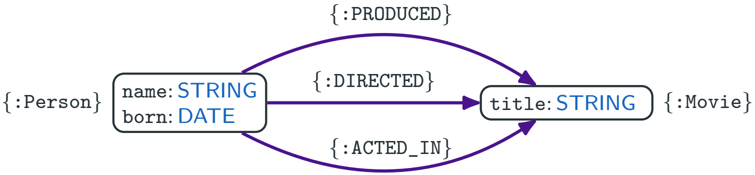
Transformation rules are openCypher scripts extended by the `GENERATE` clause

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

- `(n)` is the identifier of the (potentially!) new node *x*

DTGraph – An Example of the GENERATE clause

Input Schema



Transformation rules are openCypher scripts extended by the `GENERATE` clause

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

- `(n)` is the identifier of the (potentially!) new node x

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

- Together these rules can generate nodes which have two labels: `Actor` and `Director`

DTGraph – Another Example

Example

```
MATCH (u:User), (a:Address), (w:Location)
WHERE u.address = a.aid AND u.address = w.aid
GENERATE ((u):Person {name = u.name})-[:HasLocation]->
         ((w.countryName):Country {name=w.countryName, code=w.countryCode})
```

{:User}
u_1: name = "Jean", address = addr::a5ef

{:User}
u_2: name = "Robert", address = addr::8bc3

{:Address}
a_1: aid = addr::a5ef, cityName = "Luxemburg", cityCode = 1457

{:Address}
a_2: aid = addr::8bc3, cityName = "Luxemburg", cityCode = 1457

{:Location}
ℓ_1: aid = addr::a5ef, countryName = "Luxemburg", countryCode = LUX

{:Location}
ℓ_2: aid = addr::8bc3, countryName = "United States", countryCode = USA

DTGraph – Another Example

Example

```
MATCH (u:User), (a:Address), (w:Location)
WHERE u.address = a.aid AND u.address = w.aid
GENERATE ((u):Person {name = u.name})-[:HasLocation]->
         ((w.countryName):Country {name=w.countryName, code=w.countryCode}),
         ((u):Person {name = u.name})-[:HasAddress]->
         ((a.cityName):City {name=a.cityName, code=a.cityCode})
```

{:User}
u_1 name = "Jean"
address = addr::a5ef

{:User}
u_2 name = "Robert"
address = addr::8bc3

{:Address}
a_1 aid = addr::a5ef
cityName = "Luxemburg"
cityCode = 1457

{:Address}
a_2 aid = addr::8bc3
cityName = "Luxemburg"
cityCode = 1457

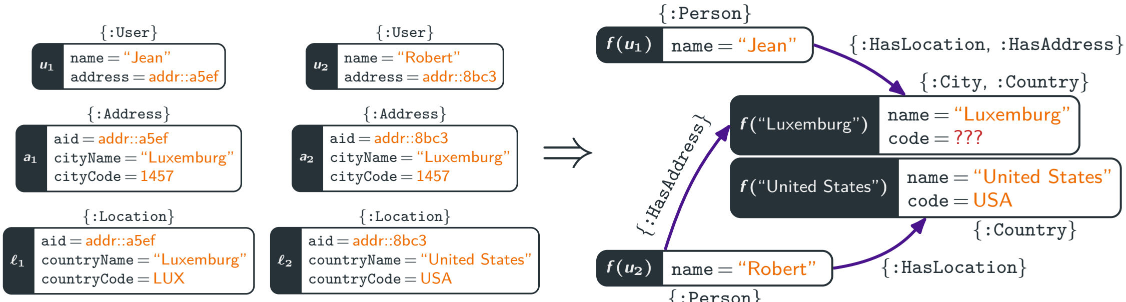
{:Location}
ℓ_1 aid = addr::a5ef
countryName = "Luxemburg"
countryCode = LUX

{:Location}
ℓ_2 aid = addr::8bc3
countryName = "United States"
countryCode = USA

DTGraph – Another Example

Example

```
MATCH (u:User), (a:Address), (w:Location)
WHERE u.address = a.aid AND u.address = w.aid
GENERATE ((u):Person {name = u.name})-[:HasLocation]->
         ((w.countryName):Country {name=w.countryName, code=w.countryCode}),
         ((u):Person {name = u.name})-[:HasAddress]->
         ((a.cityName):City {name=a.cityName, code=a.cityCode})
```



DTGraph – User Study

User Study

- 12 participants, all already familiar with openCypher

Comparison of the Ability to Understand …

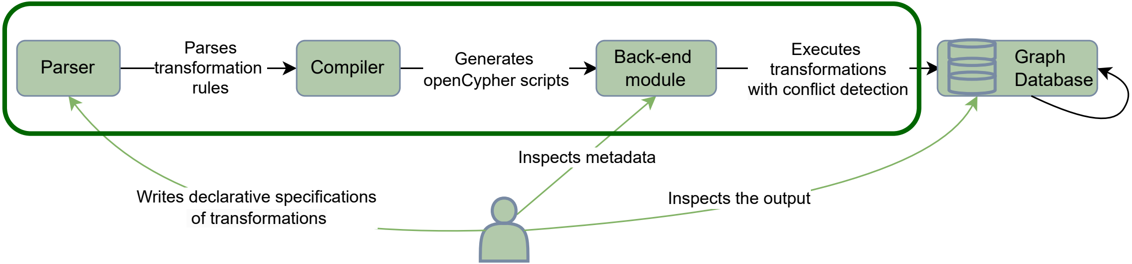
- … manual transformations with handcrafted openCypher scripts, and …
- … transformations with `GENERATE` clauses
- in clearly defined scenarios

Outcome

- Only 25% of the participants have been able to fully understand the behaviour of the openCypher scripts, whereas 67% of them succeeded with `GENERATE` clause transformations
- On average, they scored 50% on openCypher scripts and 90% on `GENERATE` clause transformations
- Participants have favoured `GENERATE` clauses by a great margin in terms of understandability, intuitiveness, and flexibility

DTGraph – System Overview

System Overview



- open-source Python3 package; Neo4j Driver
  - compatible with Neo4j and Memgraph
- Available at: `https://github.com/yannramusat/DTGraph/`

Conclusion

Several Challenges are Still Ahead of Us

- Graph-to-graph transformations (schema correspondences, schema mappings)
- Schema discovery methods leveraging ML
- Entity alignment for property graphs
- Data cleaning for property graphs
- Indexes for dynamic and streaming graphs
- Data quality for streaming graphs