PG-Keys: Keys for Property Graphs

Linked Data Benchmark Council
Property Graph Schema Working Group

SIGMOD 2021

PG-Keys team: from industry, academia, ISO, LDBC, PGSWG

Renzo Angles	Universidad de Talca, IMFD Chile
Angela Bonifati	Lyon 1 Univ., Liris CNRS & INRIA
Stefania Dumbrava	ENSIIE & Inst. Polytech. de Paris
George Fletcher	Eindhoven Univ. of Technology
Keith W. Hare	JCC Consulting Inc., Neo4j
Jan Hidders	Birkbeck, Univ. of London
Victor E. Lee	TigerGraph
Bei Li	Google
Leonid Libkin	U. of Edinburgh, ENS-Paris/PSL, Neo4j
Wim Martens	University of Bayreuth
Filip Murlak	University of Warsaw
Josh Perryman	Interos Inc.
Ognjen Savković	Free Univ. of Bozen-Bolzano
Michael Schmidt	Amazon Web Services
Juan Sequeda	data.world
Sławek Staworko	U. Lille, INRIA LINKS, CRIStAL CNRS
Dominik Tomaszuk	Inst. of Comp. Sci., U. of Bialystok

LDBC member companies and institutions

ANT GROUP

ORACLE LABS

ArangoDB

intel

aws

CHOROGRAPH

MEM GRAPH

KATANA GRAPH

JCC Consulting, Inc.

TigerGraph

*Sparsity

1794 ENS ÉCOLE NORMALE SUPÉRIEURE

ontotext

neo4j

Birkbeck UNIVERSITY OF LONDON

FORTH

THE UNIVERSITY of EDINBURGH

and many Associate Members ...

Why PG-Keys?

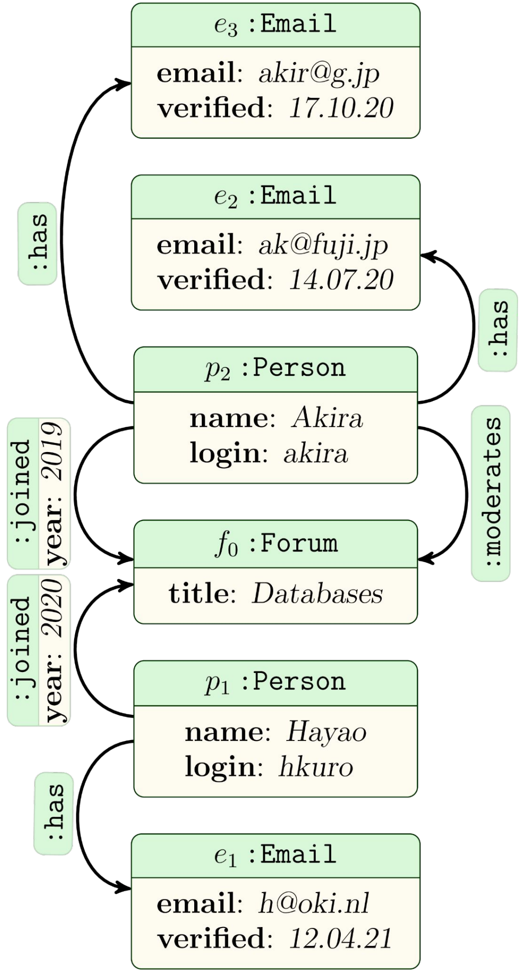
Why Property Graphs?

Graphs and graph analytics are ubiquitous

- Social networks
- Bio- and chemical networks
- Logistics, transportation, smart cities
- Communication networks
- Financial networks
- Crime networks
- Knowledge graphs
- …

Property graph model is upcoming standard

- Node- and edge-labeled directed graph
- Nodes and edges have properties
  - i.e., attribute-value pairs



Why Keys for Property Graphs?

Keys are … key in data management for identifying, referencing and constraining objects.

For example, `Person` nodes

- are uniquely identified by their login ID
- can be referenced using one of their email addresses (and it is mandatory that each person has at least one email), of which at most one can be the preferred email.
- have zero or more aliases which are exclusive (i.e., no two people can share an alias)

and discussion `Forum` nodes are identified by the forum's name and the person who moderates the forum

- (:Person)<-[:hasModerator]-(:Forum)

Why Keys for Property Graphs?

Keys are vital for managing both **identity** and **integrity** in the graph

- Example: for an integrated knowledge graph, modeling key constraints in underlying data sources and in maintaining data quality and consistency in the graph itself.

Why Now?

Because property graphs technologies are being standardized now in ISO

- LDBC and its PGSWG is a community of researchers, practitioners, and database vendors working towards consensus recommendations informing this process

and, furthermore, system vendors are moving ahead in diverse ways ...

Current Graph DB Support for Keys is Limited

Landscape is *diverse*:

- Some systems offer property-based primary keys for nodes
- Some systems support uniqueness
- Some systems support mandatoriness

Yet we need to support all of these, and more, to satisfy current practical needs.

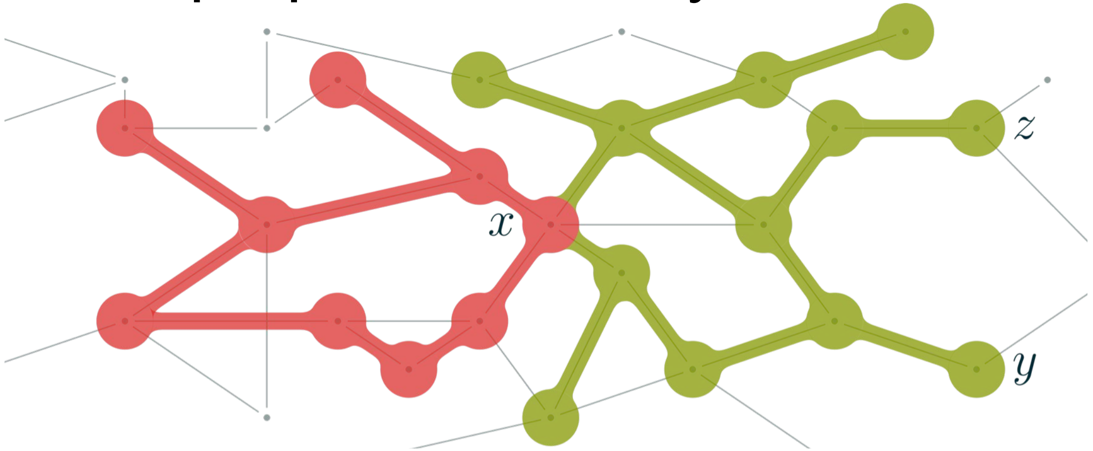
There is already a *significant drift* between database vendors

We need to *get on the same page*

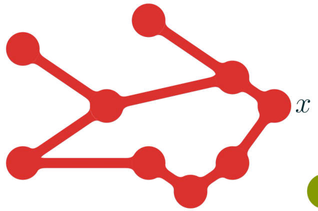
We need to *bring the best of academic work to the needs of industry*

Guided tour of PG-Keys

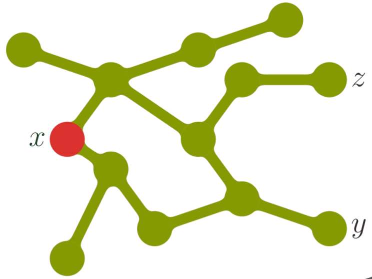
Our proposal: PG-Keys



```
FOR x WITHIN
```



```
IDENTIFIER y, z WITHIN
```



Design requirements

1. Flexible choice of key scope and descriptor of key values.
2. Keys for nodes, edges, and properties.
3. Identify, reference, and constrain objects.
4. Easy to validate.

Flexible choice of scope and key values

Declaratively specify the scope of the key and its values in your favourite PG query language (a parameter of PG-Keys). Here we use Cypher-like syntax.

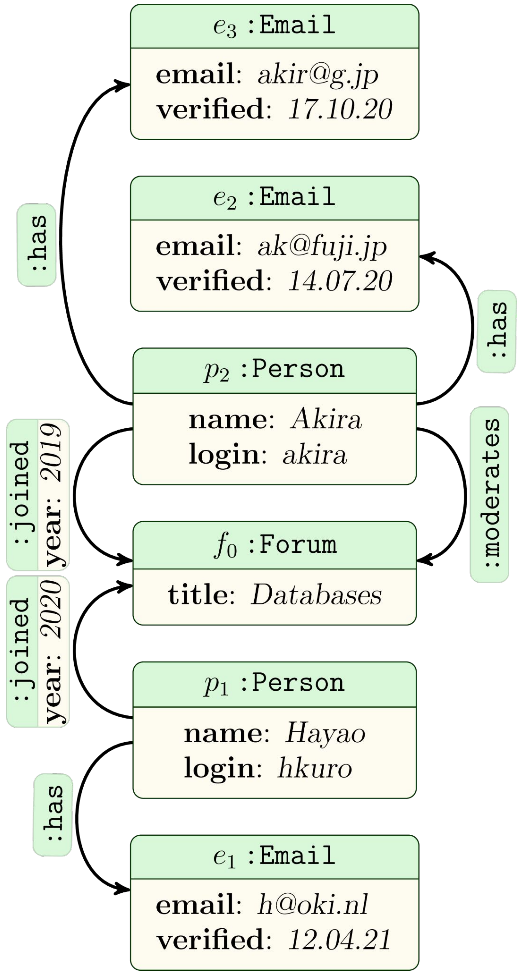
For instance

```
FOR p WITHIN (p:Person) IDENTIFIER p.login;
```

says that “each person is identified by their login”, and

```
FOR f WITHIN (f:Forum)<-[:joined]-(:Person)
IDENTIFIER f.name, p WITHIN (f)<-[:moderates]-(p:Person);
```

says that “each forum with a member is identified by its name and moderator”.



Keys for nodes, edges, and properties

The scope query selects a set of nodes, edges, or property values.

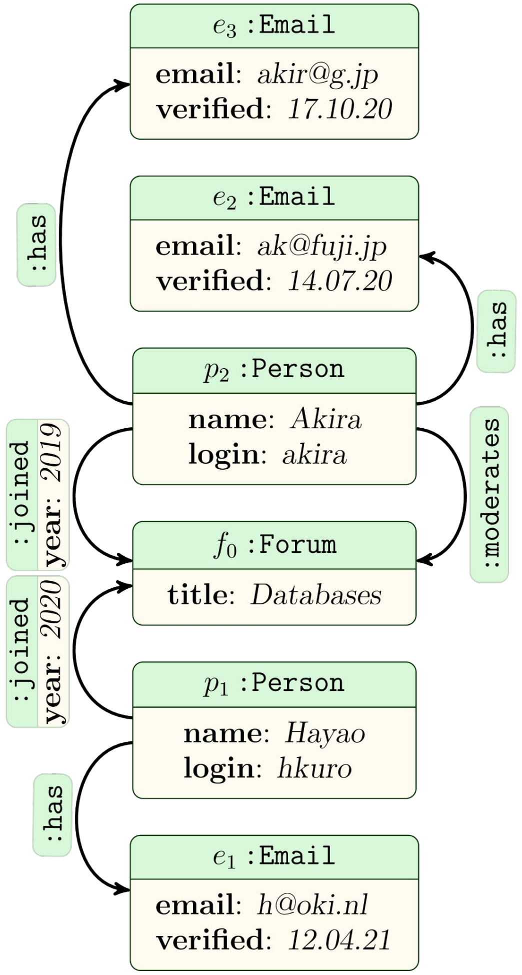
For instance,

```
FOR p WITHIN (p:Person) IDENTIFIER p.login;
```

says that "each `Person` node is identified by the value of property `login`", and

```
FOR e WITHIN (:Person)-[e:joined]->(:Forum)
IDENTIFIER p,f WITHIN (p:Person)-[e:joined]->(f:Forum);
```

says that "each `joined` edge is identified by its endpoints (i.e., no other `joined` edge has the same endpoints, so one cannot join the same forum twice)".



Identify, reference, and constrain objects

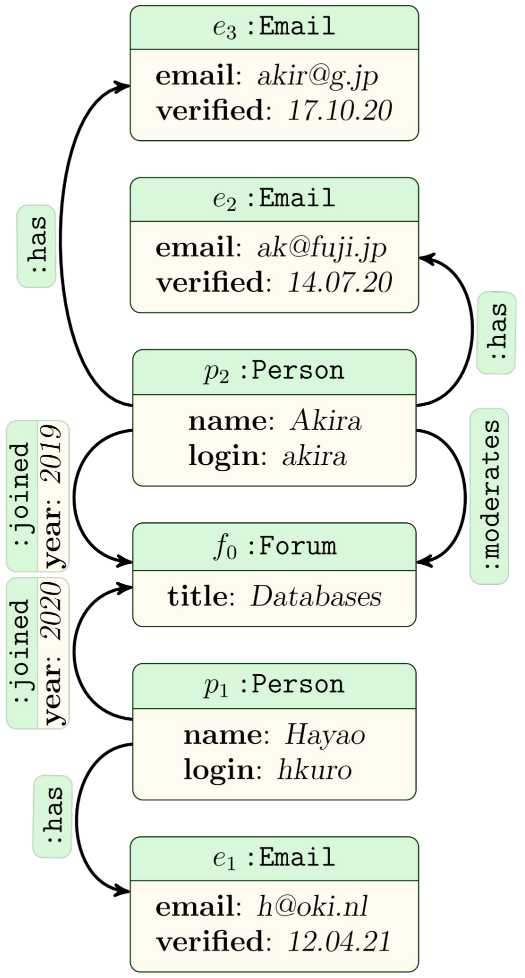
Identification is provided by `IDENTIFIER`:

```
FOR f WITHIN (f:Forum)<-[:joined]-(:Person)
IDENTIFIER f.name, p WITHIN (f)<-[:moderates]-(p:Person)
```

`IDENTIFIER` means:

- `EXCLUSIVE` - no objects in the scope share a key value;
- `MANDATORY` - each object in the scope has at least one key value;
- `SINGLETON` - each object in the scope has at most one key value.

In SQL, `EXCLUSIVE` is `UNIQUE`, `MANDATORY` is `NOT NULL`, and `SINGLETON` is always ensured by 1NF. In property graphs, all three are needed.



Identify, reference, and constrain objects

Referencing is provided by `EXCLUSIVE MANDATORY`:

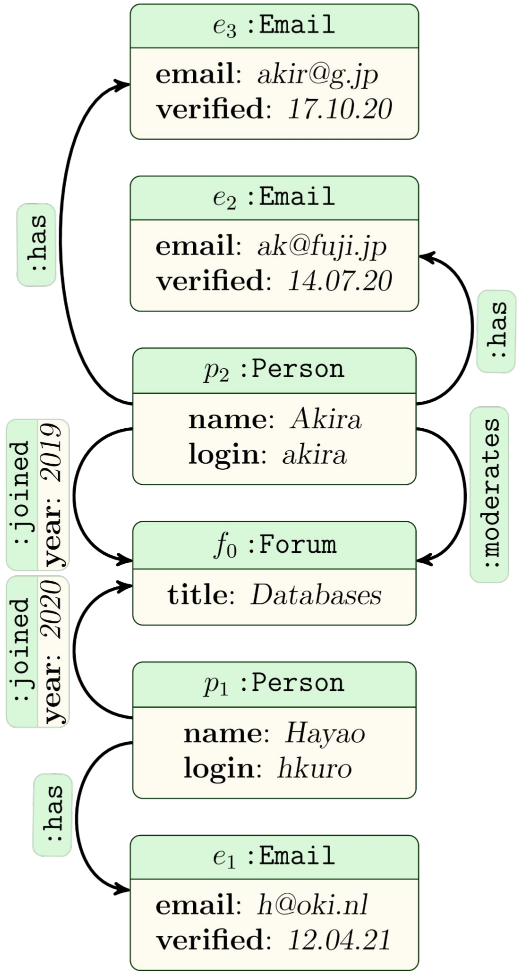
```
FOR p WITHIN (p:Person)
EXCLUSIVE MANDATORY e WITHIN (p)-[:has]->(e:Email);
```

That is, “emails are not shared and each person has an email”.

Constraining can be done in many ways. For example,

```
FOR p WITHIN (p:Person)
EXCLUSIVE p.alias;
```

```
FOR p WITHIN (p:Person)
EXCLUSIVE SINGLETON e WITHIN (p)-[:preferred]->(e:Email);
```



Easy to validate

To check that a PG-Key holds, we can run queries to find violations.

For instance,

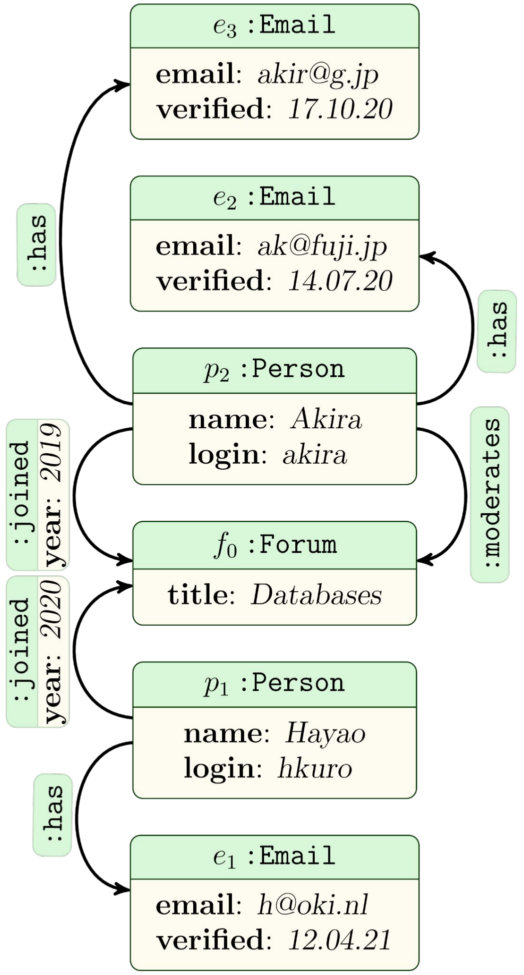
```
FOR p WITHIN (p:Person)
EXCLUSIVE MANDATORY e WITHIN (p)-[:has]->(e:Email);
```

holds if both queries below return nothing:

```
MATCH (p1:Person)-[:has]->(:Email)<-[:has]-(p2:Person)
WHERE p1 <> p2 RETURN p1, p2;
```

```
MATCH (p:Person)
WHERE NOT EXISTS (p1:Person)-[:has]->(:Email);
```

Incremental validation or batching will require additional mechanisms.



Extending PG-Keys

Null Values

If the data can have null values, then we need to know what

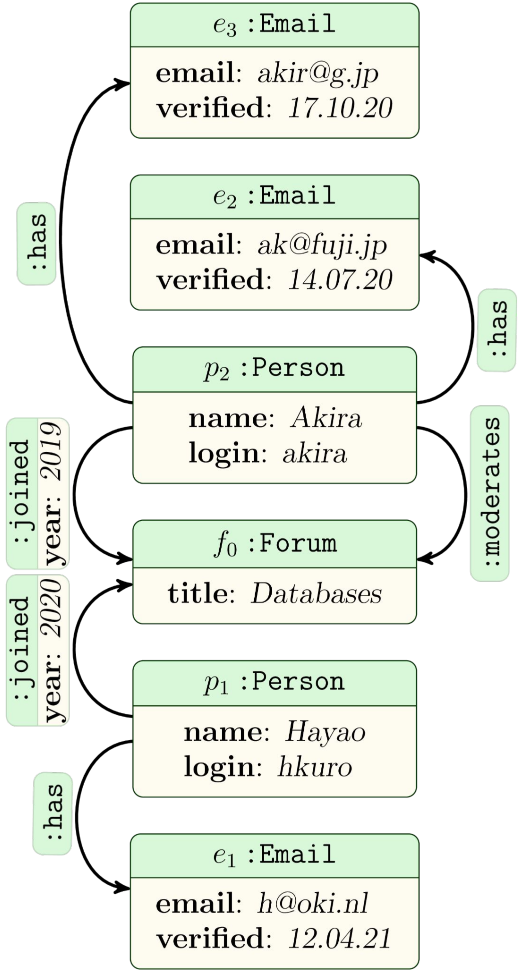
```
FOR p WITHIN (p:Person) WHERE p.age > 30
...;
```

means if `p.age` can be NULL.

Our approach:

Design PG-Keys such that one can validate a key K by executing a query Q_K that finds violations of K

We thought everything through again to see to which extent this goal can be reached; essentially, enforce “`p.age > 30 AND p.age IS NOT NULL`”



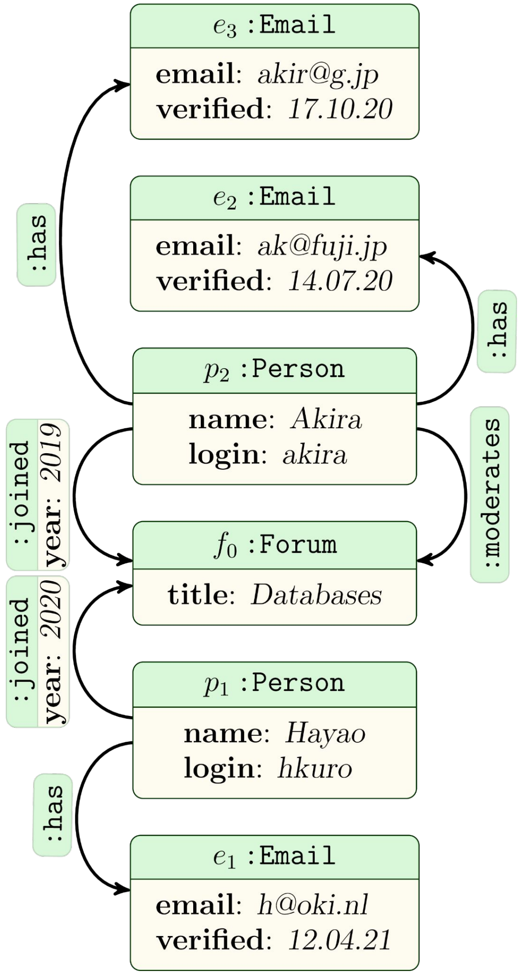
Regular Path Queries

If posts in forums can reply to other posts, then

```
FOR x WITHIN (x:Post)
IDENTIFIER p WITHIN
        p = (x)-[:replyTo*]->()->[:OP]->(f:Forum);
```

means that

> “Each post is identified by its reply-to path in the forum where it was posted”



Complex Values

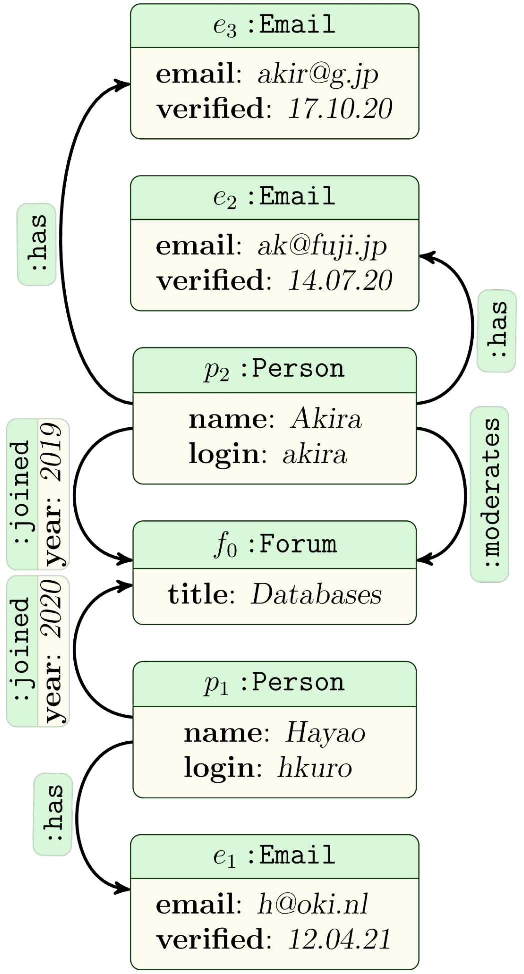
Our data model already allows complex values such as

tuples sets lists JSON structures

In order to use these in PG-Keys, we leverage the query language:

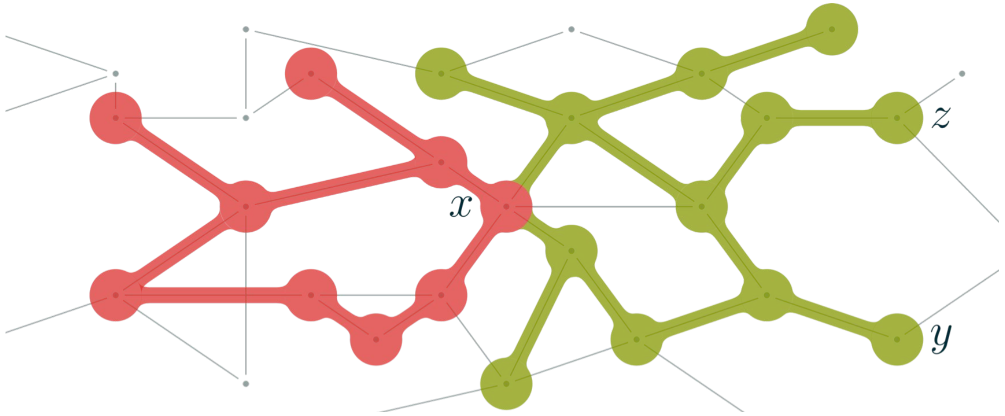
```
FOR p WITHIN (p:Person)
EXCLUSIVE MANDATORY
    p.phone[@.category='official'].number;
```

- "Each person has an official telephone number" and
- "No two persons have the same official telephone number"



Looking Ahead

Looking ahead



- PG-Keys is informing the design of ISO’s new GQL query language via the LDBC liaison
- PG-Keys is a call to action also to industry and academia
  - Guide the design and engineering of commercial and non-commercial graph systems and solutions
  - Open problems for research include:
    - Validation and maintenance complexity for specific query languages
    - Implication and inference problems
    - Static analysis for optimization purposes
    - Richer PG constraint languages
- And PGSWG is just getting started!
  - Design of PG schema and constraint languages
  - Extensions to the PG model driven by practical applications, e.g., meta-properties.