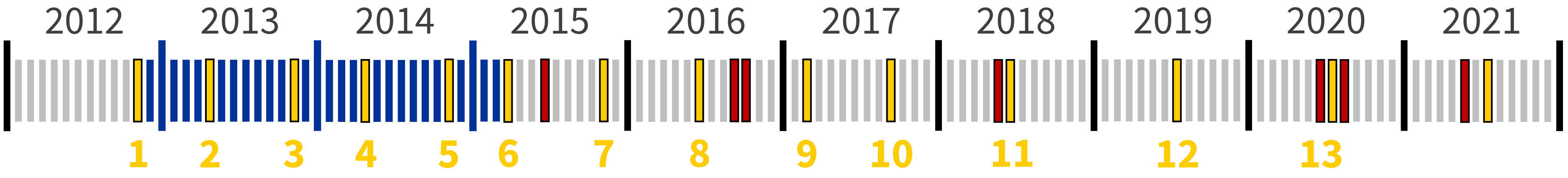
Property Graph Schema Working Group: Where we are today, and what’s next

Jan Hidders (Birkbeck, University of London)
Juan Sequeda (data.world)

LDBC project, benchmark papers & meetings



SNB Interactive
SIGMOD 2015

Graphalytics
VLDB 2016

SNB BI draft
GRADES 2018

Datagen + deletions
GRADES 2020

SPB
BLINK 2016

ACID tests
TPCTC 2020

EU FP7 project

TUC meetings

Benchmark papers

LDBC: member companies and institutes

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
redis

2014: LDBC Graph Query Language WG

LDBC starts working group

Industry: Neo4j, SAP, Oracle, Capsenta

Academia: Univ. de Talca, PUC Chile, Univ. de Chile, CWI Amsterdam, TU Eindhoven, TU Dresden

2017: G-CORE

Results in proposal for core language: G-CORE

https://arxiv.org/pdf/1712.01550.pdf

Published in SIGMOD 2018

Regarded as useful exercise by all sides

July 2018: Shonan Seminar

Shonan Seminar: Graph Database Systems

GQL Community started, continue G-CORE work

More open community, also covers data model

March 2019: W3C workshop

W3C workshop on graph data -- Creating

Bridge: RDF, Property Graph, and SQL, Berlin

PGSWG started

September 2019: WG3 starts GQL

ISO/IEC JTC 1/SC 32/WG3 gets backing from

ISO/IEC to start on GQL and SQL/PGQ:

(1) SQL/PGQ : extension of SQL

(2) GQL : stand-alone query language

September 2020: Formal link LDBC and WG3

LDBC Liaisons attend WG3 meetings

GQL comm. and WG3 can exchange documents

LDBC membership is now required to participate

DISCLAIMER

- Community Group: group of people from industry and academia who are interested in graphs
- Our work is only advisory: WG3 is not bound by our advice
- Possible deliverables
  - Recommendations to WG3
  - Academic collaboration
  - Academic-Industry collaboration
  - Open source software
- We do not always aim for consensus
- Anyone can join!

GQL Community WGs

ELWG

Existing Languages

Surveys existing query languages

PGSWG

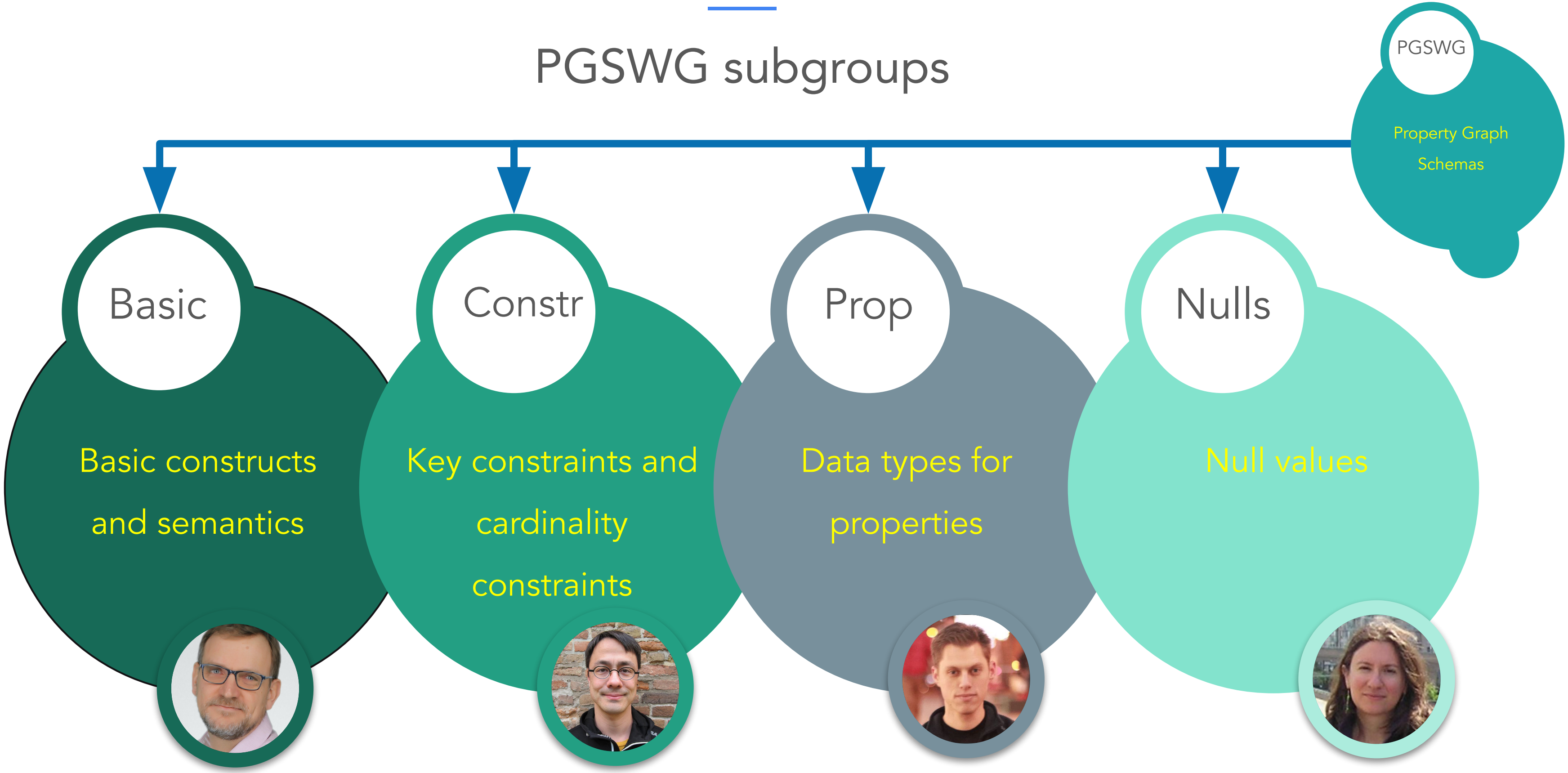
Property Graph Schemas Syntax and semantics of schema language

FSWG

Formal Semantics

Formal semantics of GQL

PGSWG subgroups



PGSWG

Property Graph Schemas

Basic

Basic constructs and semantics

Constr

Key constraints and cardinality constraints

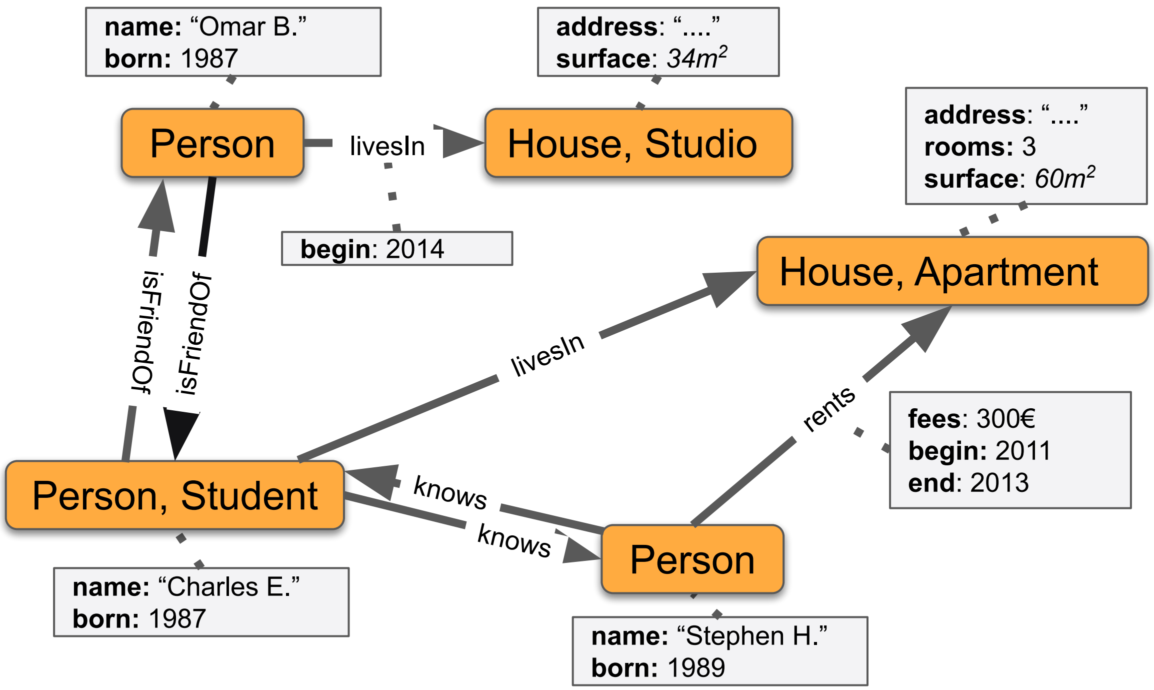
Prop

Data types for properties

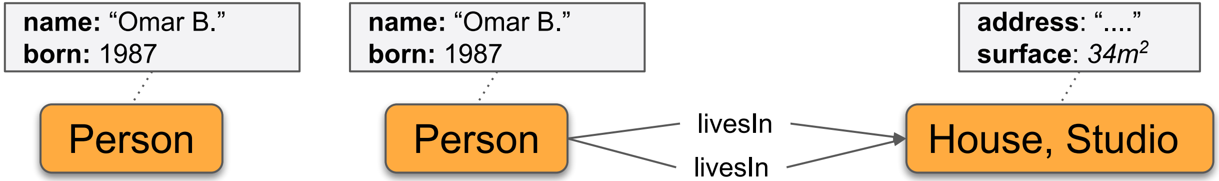
Nulls

Null values

Property Graphs as defined by PGSWG



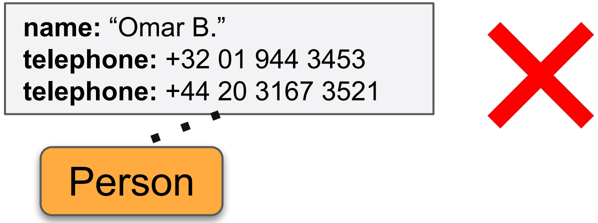
Basic Data Model Assumptions (1/4)



✓

Duplicate nodes and edges are allowed.

Basic Data Model Assumptions (2/4)

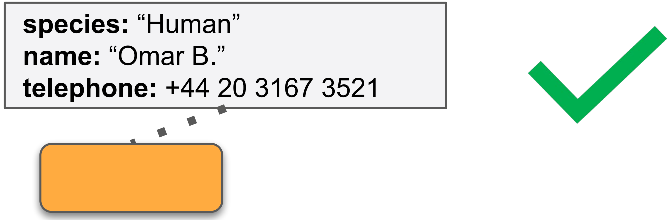


name: “Omar B.”
telephone: +32 01 944 3453
telephone: +44 20 3167 3521

Person

Properties with identical names are not allowed.

Basic Data Model Assumptions (3/4)

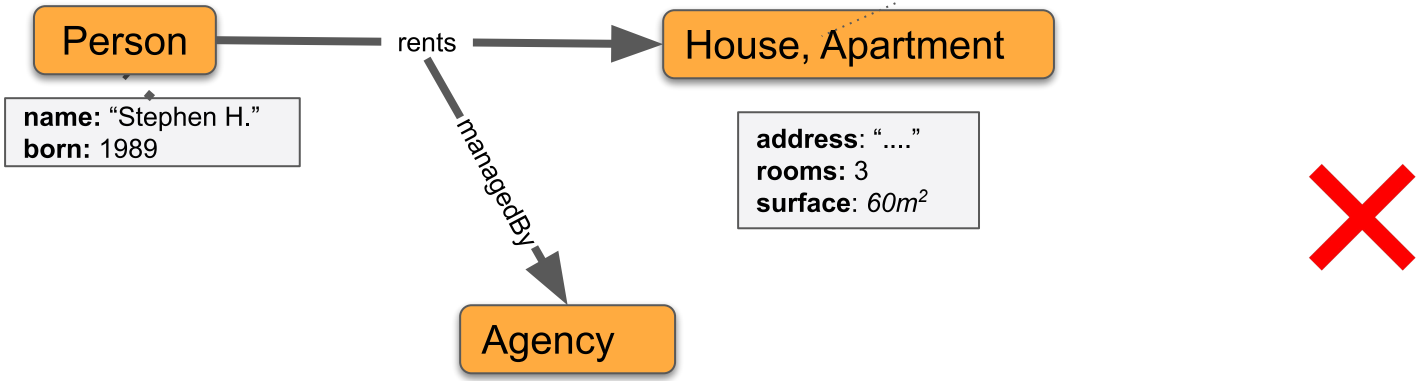


species: “Human”
name: “Omar B.”
telephone: +44 20 3167 3521

✓

Nodes and edges with **no** labels are allowed.

Basic Data Model Assumptions (4/4)



Edges only connect nodes.

Main goals

- Keep it **simple** where possible
- Give it a **formal** semantics, ideally based on a well-understood formalism
- Allow simple **light-weight** schemas as well as **full-blown** database schemas
- Cover the range from **closed** schemas (fully fixing the vocabulary) to **open** schemas (only partially or not fixing the vocabulary)
- Facilitate gradual migration between graphs with **no prescriptive** schema, via a partially prescriptive schema, to a **fully prescriptive** schema
- **Familiar** through similarity to existing conceptual data models: EER, UML Class diagrams, ORM2, …

The basic structure of a schema

A schema is a set of node types and ..

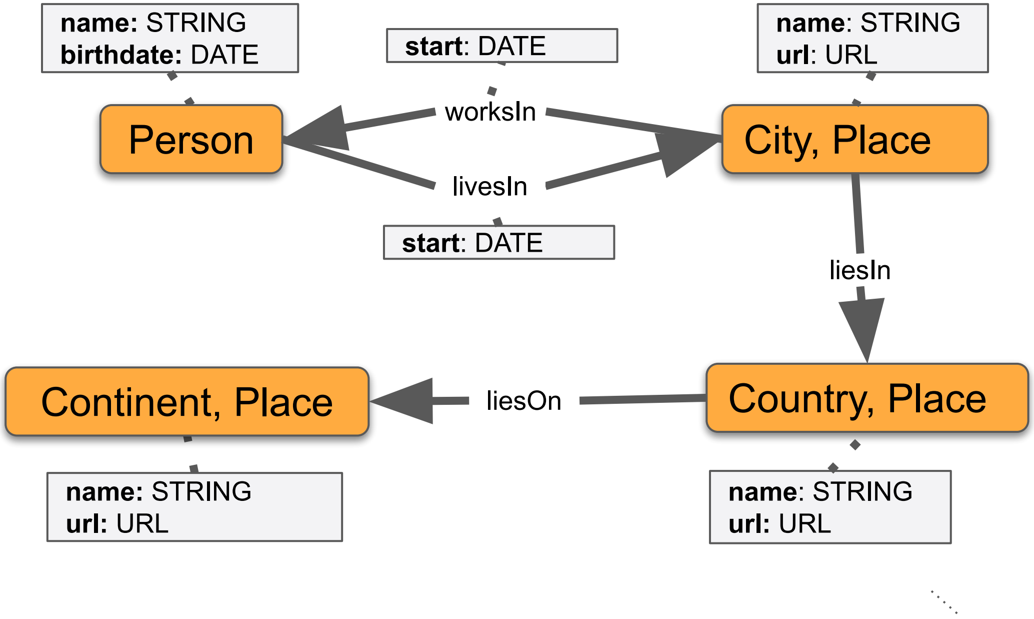
```
$person = (:Person { name::STRING, birthdate::DATE })
$city = (:City Place { name::STRING, url::URL })
$country = (:Country Place { name::STRING, url::URL })
$continent = (:Continent Place { name::STRING, url::URL })
```

.. a set of edge types

```
$livesIn = (:$person)-[:livesIn { start::DATE }]->(:$city)
$worksIn = (:$person)-[:worksIn { start::DATE }]->(:$city)
$cityLiesIn = (:$city)-[:liesIn]->(:$country)
$countryLiesOn = (:$country)-[:liesOn]->(:$continent)
```

Basic semantics: every node and edge must conform to a type.

Graph Representation of Schema



Well-advanced discussions

Prop

Data types for properties

Property Types

- Node and Edges types, Record types, collection types (array), basic types (int, varchar, etc)
- Partial alignment with SQL types
- Metaproperties: all property values and their subvalues can be annotated with meta-properties

```
$person1 = (:Person { name::STRING, birthdate::DATE })

$person2 = (:Person { name::STRING, birthdate::{
                                      day::STRING,
                                      month::STRING,
                                      year::STRING } })

$livesIn = (:$person)-[:livesIn { start::DATE }]->(:$city)
```

Metaproperties

```
$city = :City {
  name STRING,
  population INTEGER @{
    point_in_time DateTime @{ confidence_score FLOAT },
    determination_method STRING
  }
}
```

properties

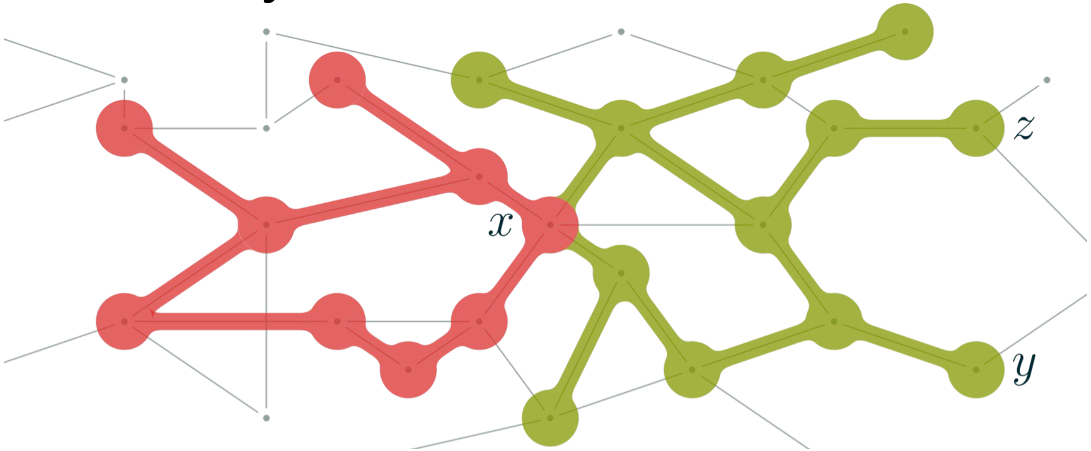
Meta-properties for ‘population’

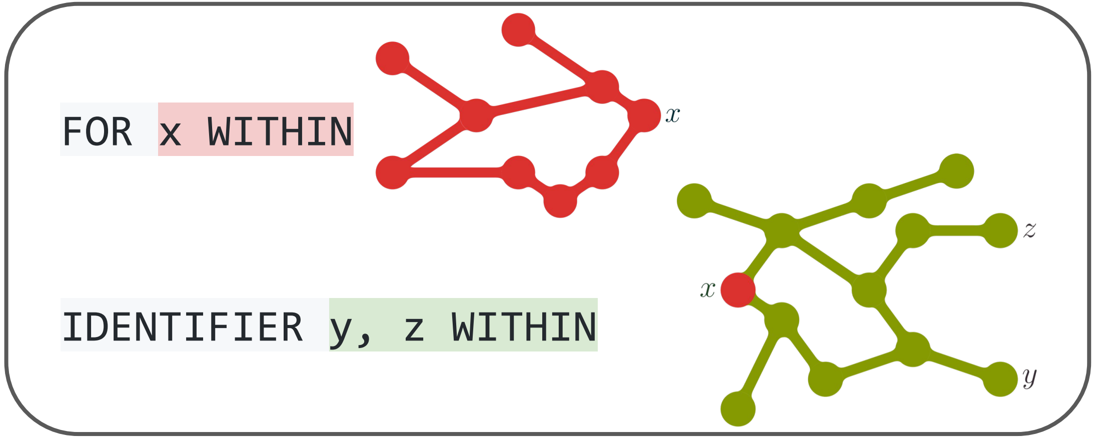
Meta-properties for ‘point_in_time’

Key constraints

- Simple key constraints: sets of properties
- Complex key constraints: nodes and edges are identified by combinations of directly or indirectly connected properties and nodes
- See paper: PG-Keys: Keys for Property Graphs. SIGMOD Conference 2021

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.

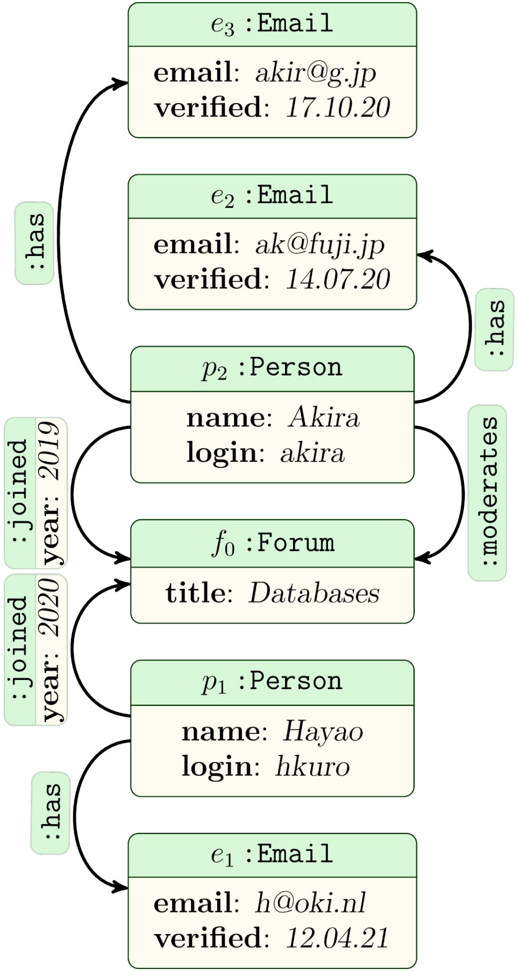
For instance

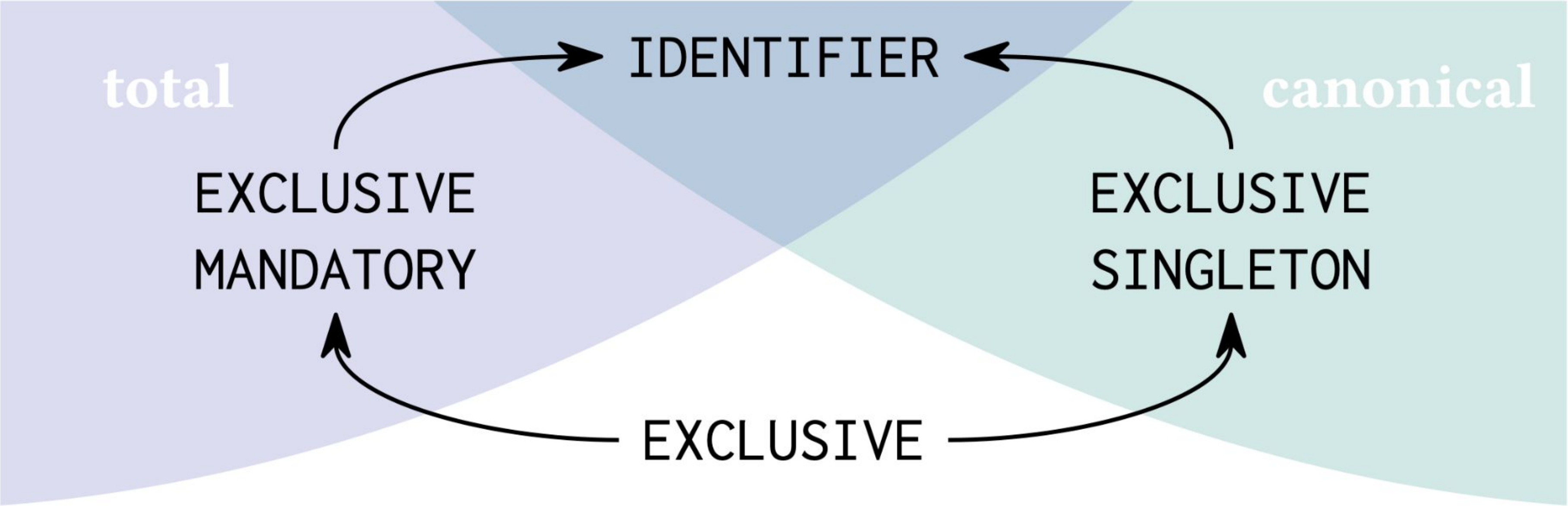
```
FOR p WITHIN (p:Person) IDENTIFIER p.login;
```

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”.





total

IDENTIFIER

canonical

EXCLUSIVE
MANDATORY

EXCLUSIVE
SINGLETON

EXCLUSIVE

Cardinality constraints

- Simple cardinality constraints: cardinality constraints on edge types (upper/lower bounds) → ER Diagrams
- Complex cardinality constraints: upper and lower bounds for results of graph patterns → Nodes in a selected graph pattern

Person must live in at least one city:

```
$livesIn = (:$person)=[:livesIn[M:1] { start::DATE }]=>(:$city)
```

Person can work in multiple cities.

```
$worksIn = (:$person)-[:worksIn[M:N] { start::DATE }]->(:$city)
```

Schema flexibility

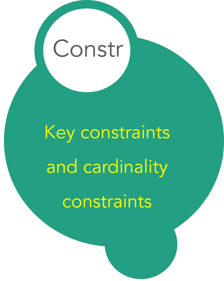
- In node, edge types we can mark properties as optional
- We can indicate that a node and edge type is open: extra properties are allowed

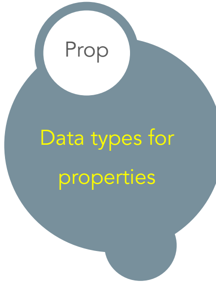
```
$person1 = (:Person { name::STRING, birthdate?::DATE })


$person2 = (:Person { name::STRING, birthdate::DATE, .. })


(:Person { name::”Juan”}) YES $person1 NO $person1


(:Person { name::”Juan”, birthdate::”17-10-1985”, email::”juan@data.world”
}) NO $person1 YES $person1
```





Overlapping types

- The chosen semantics does not allow meaningful overlap of types
- An analogue to overlapping subtypes in conceptual data models (e.g., EER and UML diagrams) has been added where types can be explicitly indicated as combinable

```
$manager = (:Manager { name::STRING})

$engineer = (:Engineer { name::STRING})

(:Manager :Engineer { name::”Juan”}) NO
```

HOWEVER...

```
$manengineer = (:Manager :Engineer { name::STRING})

(:Manager :Engineer { name::”Juan”}) YES
```

Just-started discussion

Nominalised vs structural type behavior

- Types with different names do not overlap
- SQL does this in some places
- Do we want / need it?

Union Types

- Tagged union vs untagged union
- Alternative for NULL values?
- Too powerful?
- Necessary for deriving descriptive type?

NULL value(s)

- SQL/PGQ will have to deal with them
- Covered by optional properties?
- 3-valued logic?

Derivation of descriptive schema

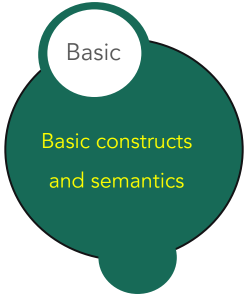
- Derive a schema if there is none
- For type inference
- For starting prescriptive schema

Basic

Basic constructs and semantics

Type inference for schemas

- Determine well-typedness of query
- Determine structure of query result



Basic

Basic constructs and semantics

Final Thoughts

Juan’s Final Thoughts

- Balance of Academia vs Industry … too many chefs in the kitchen?!?!?
  - Energy, Time Commitment
- Boiling the ocean … ok?!?!?
  - Going into deep holes?
  - Who cares about them?
- First arrive to understand the different POV before even considering consensus
- RDF Schema “done” right?
- Socio-technical Research opportunities

Jan’s Final Thoughts

- Interaction with ISO’s WG3 is very encouraging
    - But requires compromises
- Creating a community is hard, but very rewarding
    - Wide variety of (quite strong) opinions
    - Has resulted in new and productive cooperation
- Database models vs Conceptual data models
    - Graph-based data models claim to approximate conceptual data models
    - Do they?
    - Should they?

Conclusion

“*We choose to go to the Moon in this decade and do the other things, **not because they are easy, but because they are hard***” - JFK

- This is not easy
    - Humans and subjectivity
    - Judgement calls
    - Balance between programming and database world
- We are reusing existing established ideas (i.e. not reinventing the wheel)
- Open up the closed world of ISO standards

THANK YOU!