Introducing PG-Schema

Schemas for Property Graphs

Who are we?

Renzo ANGLES, Universidad de Talca, Chile
Angela BONIFATI, Lyon 1 University & CNRS, France
Stefania DUMBRAVA, ENSIIE & Institut Polytechnique de Paris, France
George FLETCHER, Eindhoven University of Technology, Netherlands
Alastair GREEN, LDBC, UK
Jan HIDDERS, Birkbeck, University of London, UK
Bei LI, Google, USA
Leonid LIBKIN, University of Edinburgh, UK; RelationalAI & ENS, PSL University, France
Victor MARSAULT, Université Gustave Eiffel & CNRS, France
Wim MARTENS, University of Bayreuth, Germany
Filip MURLAK, University of Warsaw, Poland
Stefan PLANTIKOW, Neo4j, Germany
Ognjen SAVKOVIĆ, Free University of Bozen-Bolzano, Italy
Michael SCHMIDT, Amazon Web Services, USA
Juan SEQUEDA, data.world, USA
Sławek STAWORKO, RelationalAI, USA; University of Lille, France
Dominik TOMASZUK, University of Bialystok, Poland
Hannes VOIGT, Neo4j, Germany
Domagoj VRGOČ, University of Zagreb, Croatia; PUC Chile, Chile
Mingxi WU, TigerGraph, USA
Dušan ŽIVKOVIĆ, Integral Data Solutions, UK

Different ways to use schemas

No Schema

Flexible Schema

- rapid development in early stages
- schema comes with data
- **descriptive role:** tell users & systems what to expect in the data

Partial Schema

- advanced stages of development
- **prescriptive schema** over stable data
- **descriptive schema** for stable and evolving data

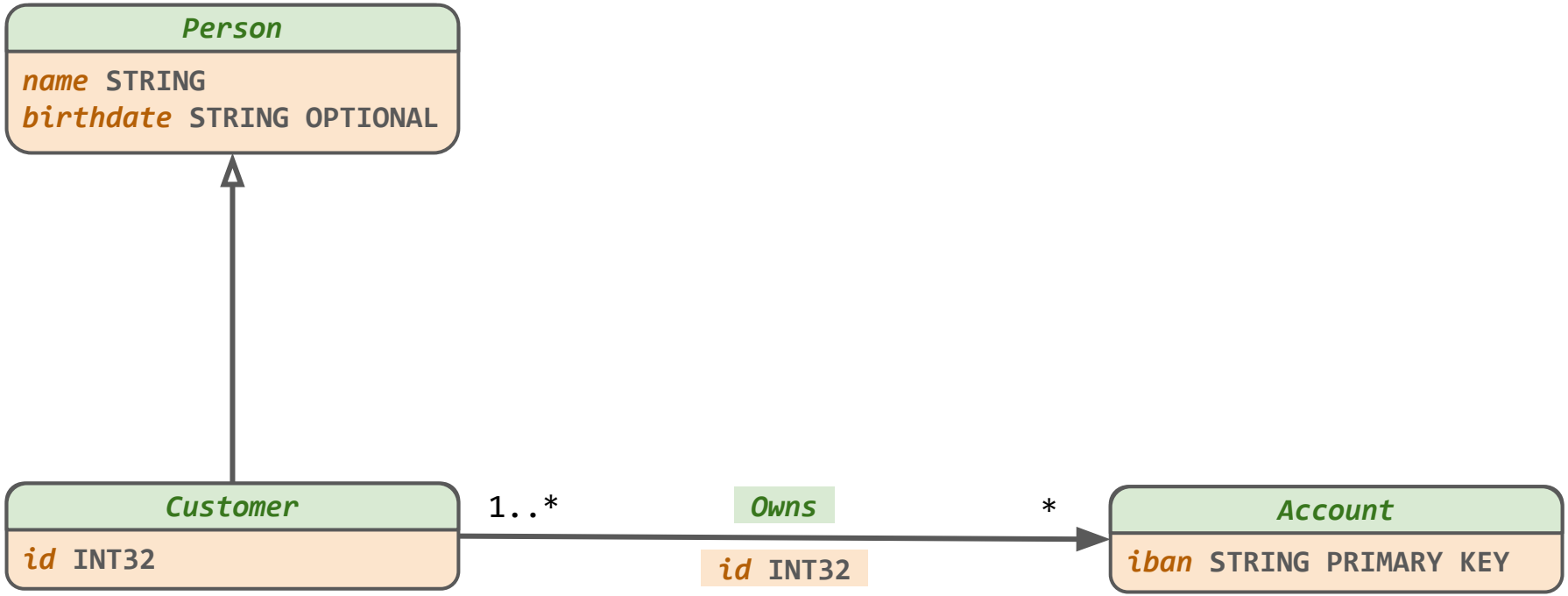
Schema First

- production settings of stable systems
- schema provided during setup
- **prescriptive role:** limit data modifications

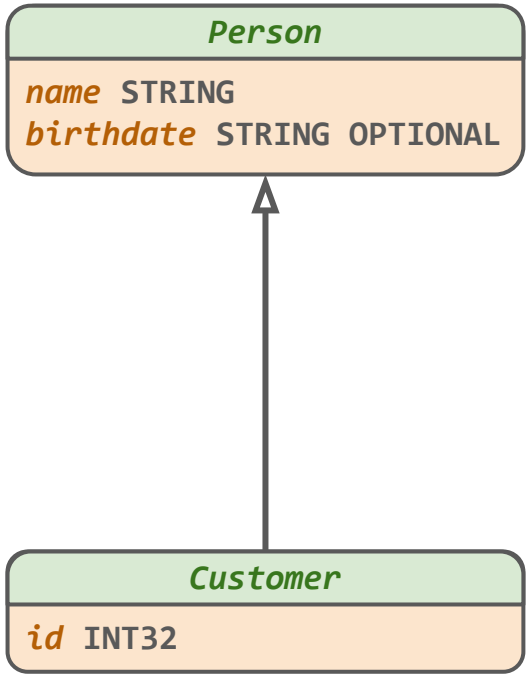
data exploration ● data visualisation ● query formulation
data transformations ● data integration
data curation ● query optimization

Ingredients of PG-Schema

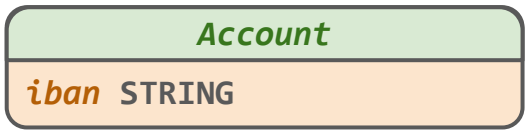
Example


Person
name STRING
birthdate STRING OPTIONAL
Customer
id INT32
1..*
Owns
id INT32
*
Account
iban STRING PRIMARY KEY

Node types

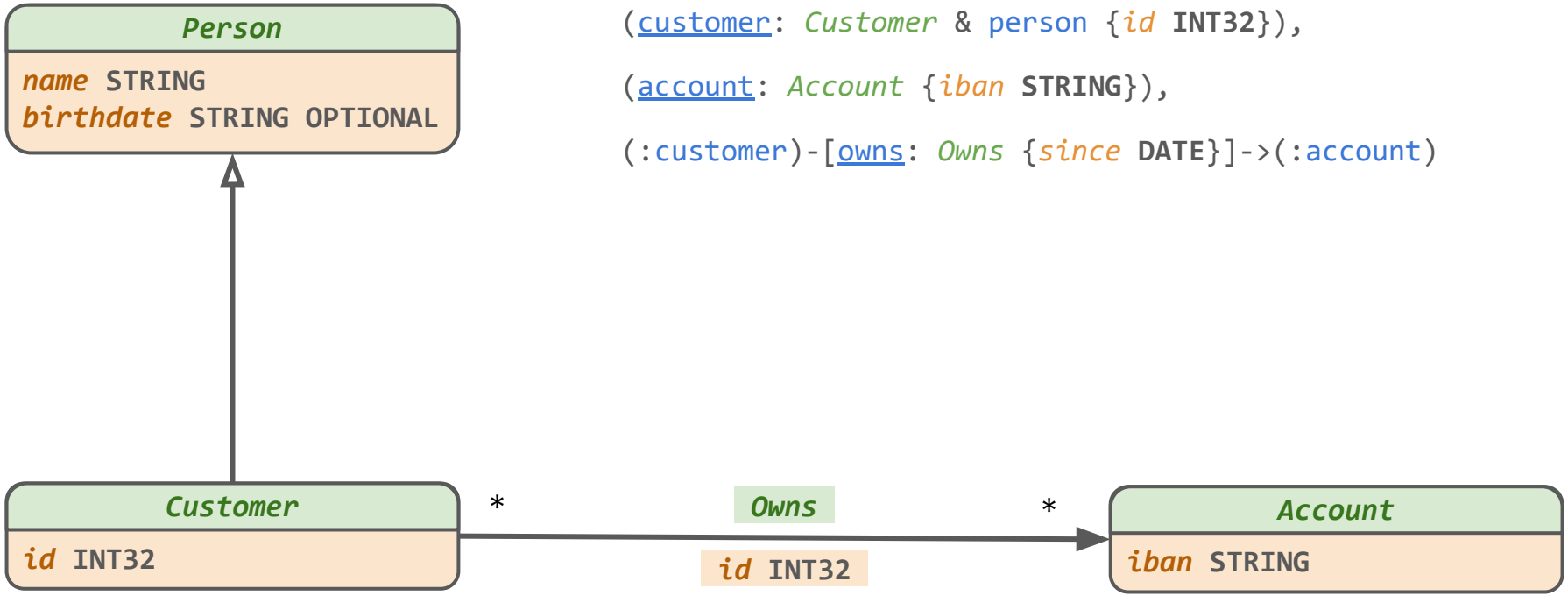



```
(person: Person {name STRING, OPTIONAL birthdate DATE}),
(customer: Customer & person {id INT32}),
(account: Account {iban STRING})
```

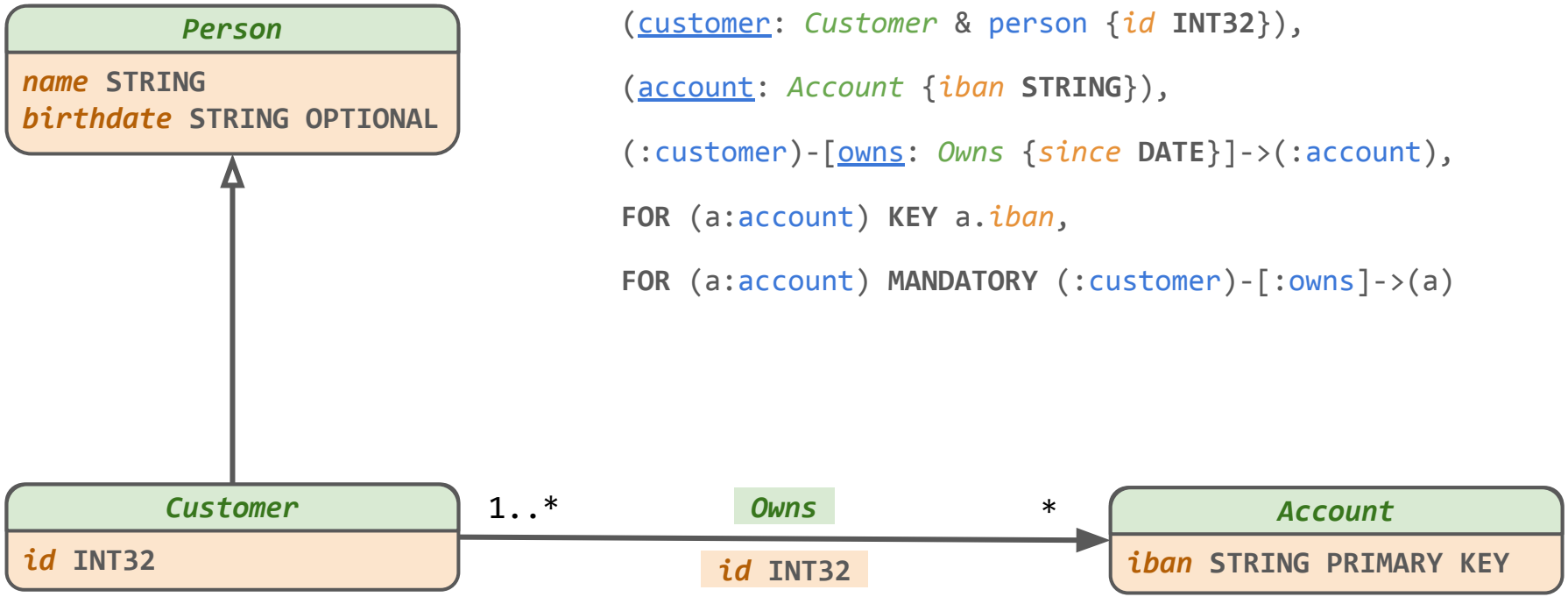


Edge types

```
(person: Person {name STRING, OPTIONAL birthdate DATE}),
(customer: Customer & person {id INT32}),
(account: Account {iban STRING}),
(:customer)-[owns: Owns {since DATE}]->(:account)
```



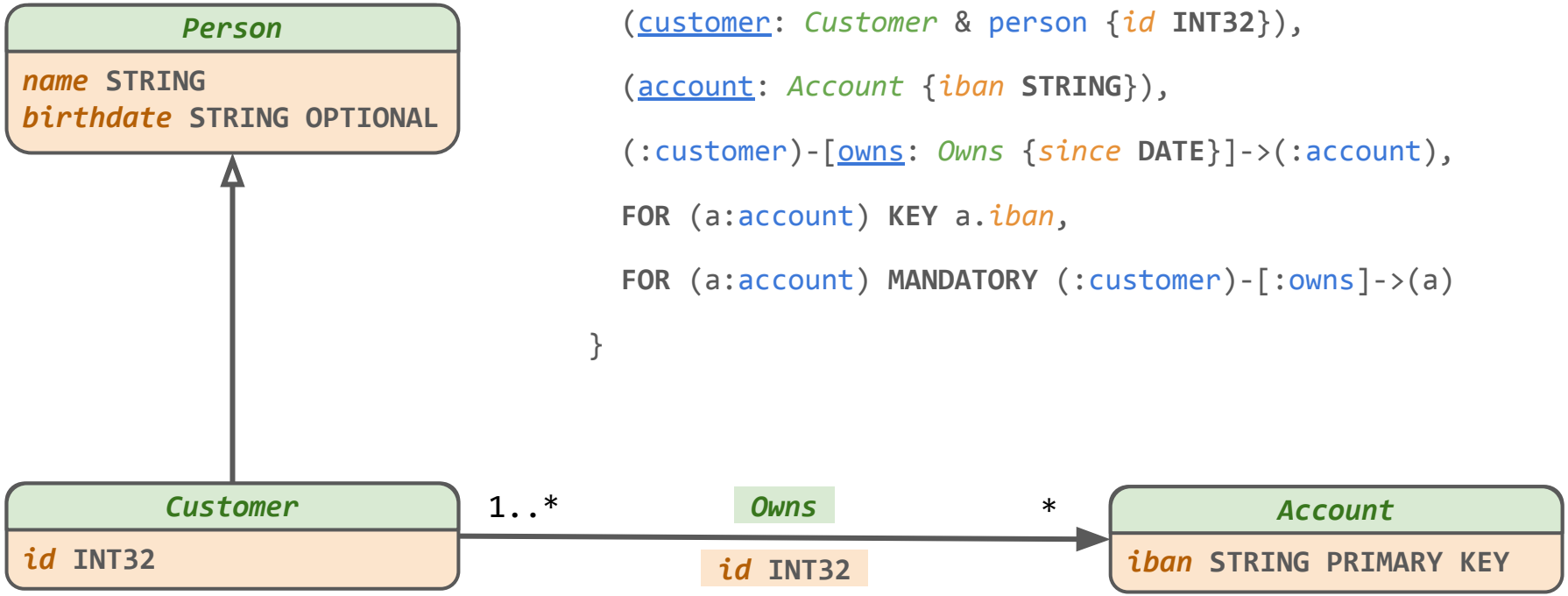
Constraints




```
(person: Person {name STRING, OPTIONAL birthdate DATE}),
(customer: Customer & person {id INT32}),
(account: Account {iban STRING}),
(:customer)-[owns: Owns {since DATE}]->(:account),
FOR (a:account) KEY a.iban,
FOR (a:account) MANDATORY (:customer)-[:owns]->(a)
```

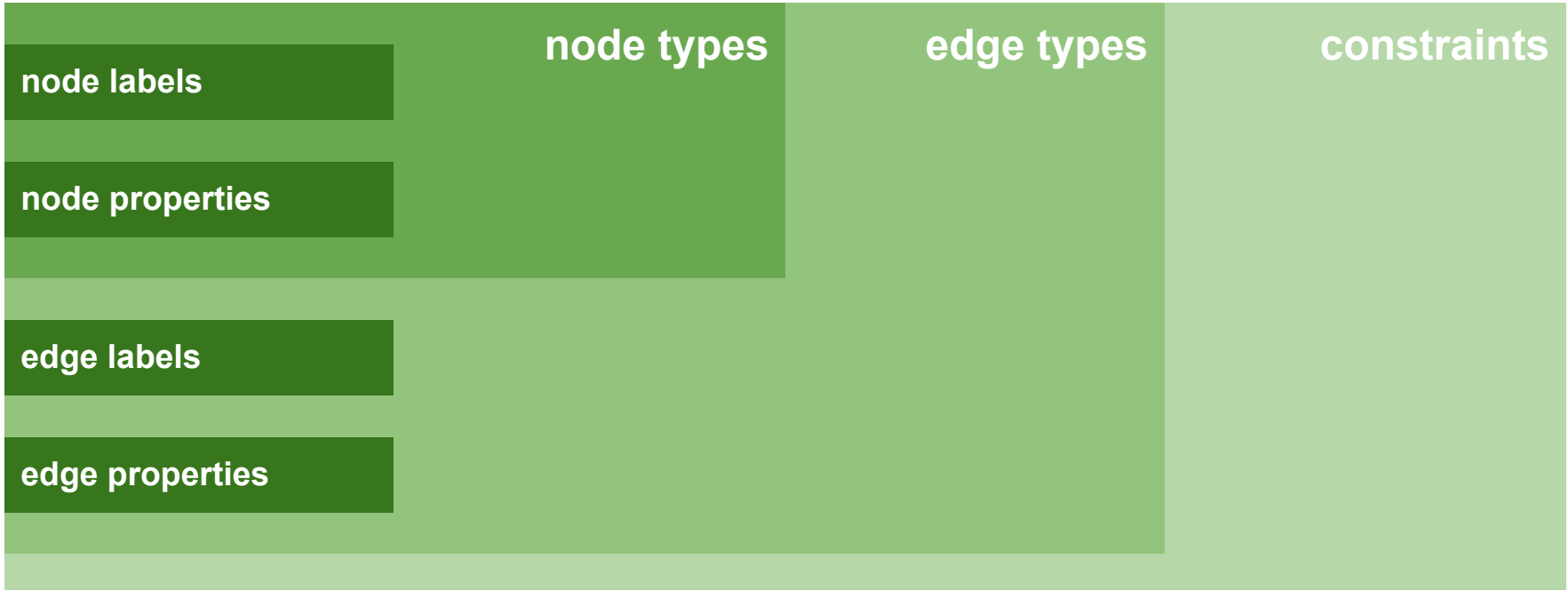
Complete PG-Schema

```
CREATE GRAPH TYPE customerGraph STRICT {
  (person: Person {name STRING, OPTIONAL birthdate DATE}),
  (customer: Customer & person {id INT32}),
  (account: Account {iban STRING}),
  (:customer)-[owns: Owns {since DATE}]->(:account),
  FOR (a:account) KEY a.iban,
  FOR (a:account) MANDATORY (:customer)-[:owns]->(a)
}
```



Superpowers of PG-Schema

Simplicity. One-way information flow



easy to understand, validate, and generate ● facilitates partial validation

Power. Compositionality

Union, intersection, and abstract types, for inheritance and more.

```
CREATE GRAPH TYPE customerGraph STRICT {
  (person: Person {name STRING, OPTIONAL birthdate DATE}),
  (company: Company {name STRING}),
  ABSTRACT (taxpayer: {taxPayerNumber STRING}),
  (customer: (person|company) & taxpayer & Customer {id INT32})
}
```

reusability ● conciseness ● modelling power

Versatility. Strict and loose schemas

STRICT schemas: elements must belong to at least one type and constraints must hold.

```
CREATE GRAPH TYPE customerGraph STRICT {
  (person: Person {name STRING, OPTIONAL birthdate DATE}),
  (customer: Customer & person {id INT32}),
  (account: Account {iban STRING}),
  (:customer)-[owns: Owns {since DATE}]->(:account),
  FOR (a:account) KEY a.iban,
  FOR (a:account) MANDATORY (:customer)-[:owns]->(a)
}
```

schema first ● partial schema ● flexible schema

Versatility. Strict and loose schemas

LOOSE schemas: elements may belong to zero types, but the constraints must hold.

```
CREATE GRAPH TYPE customerGraph LOOSE {
  (person: Person {name STRING, OPTIONAL birthdate DATE}),
  (customer: Customer & person {id INT32}),
  (account: Account {iban STRING}),
  (:customer)-[owns: Owns {since DATE}]->(:account),
  FOR (a:account) KEY a.iban,
  FOR (a:account) MANDATORY (:customer)-[:owns]->(a)
}
```

schema first ● partial schema ● flexible schema

Versatility. Closed and open types

CLOSED types (default) allow only explicitly mentioned or inherited labels and properties.

OPEN types allow arbitrary additional labels and properties.

```
CREATE GRAPH TYPE customerGraph STRICT {
  (person: Person OPEN {name STRING, OPTIONAL birthdate DATE}),
  (customer: Customer & person {id INT32, OPEN })
}
```

schema first ● partial schema ● flexible schema

Also in the paper

Also in the paper

- Systematic analysis of design requirements.
- Full grammar of PG-Schema (excluding PG-Keys).
- Formal semantics of PG-Schema (excluding PG-Keys).
- Detailed comparison with existing schema formalisms.
- Possible extensions.

Takeaway

For industry

- **PG-Schema is a simple, yet powerful and versatile schema language for property graphs, rooted in the experience of 30+ researchers, engineers, and standards contributors.**
- By implementing it in your system you will increase functionality to better support current and future customer demands.
- If full PG-Schema seems too much, talk to us about an adequate fragment.

For academia

- **The hard part of schema language design is striking the right balance between simplicity and power. Can we add negation or recursion?**
- Schema validation and basic schema generation is tractable, but practical maintenance algorithms are needed (incremental and batch setting).
- Powerful type compositions make visualizing schemas challenging.
- Graph data transformations and query optimization can build on PG-Schema.

Thank You