192.161 Management of Graph Data

(4.0 VU / 6.0 ECTS)

2025W

Constraints & Data Construction for Graph Data

Katja Hose

Maxime Jakubowski

mogda@list.tuwien.ac.at

Overview

1. Continuing with PG-Keys
2. Graph Construction
3. Project Specifics

PG-Keys

Adding constraints

Reminder: PG-Schema

- The strongest proposal for a schema language for PGs
- Not (yet) part of the GQL ISO standard, but working towards it
- Consists of two parts:
  - **PG-Types**: constructing types for nodes and edges
    - Type: for grouping elements that represent the same kind of object in the real world
  - **PG-Keys**: writing constraints over the graph data
    - Constraint: a closed formula that imposes limitations to the graph structure

source: dl.acm.org/doi/10.1145/3589778

Key Constraints for Property Graphs

- In relational databases, constraints play an essential role: you cannot insert data violating the constraints.
  - Primary key constraints
  - Participation constraints
- In graphs, constraints are typically *checked after the data is already there*
- Keys are for *identifying, referencing and constraining* objects
- They are <u>core components</u> of PG-Schema

Example: Person Nodes

- <u>*Uniquely identified*</u> by their login ID
- <u>*Referenced*</u> using one of their email addresses
  (+ having an email address is mandatory)
- <u>*At most one*</u> can be a preferred email

Example: Forum Nodes

- <u>*Uniquely identified*</u> by their name and the moderator node
  `(: Person)<-[:hasModerator]-(:Forum)`
- Not a property-based primary key: identity depends on other nodes and edges

Key Scope and Descriptor

A ***key*** consists of:

- a ***scope***
- a ***descriptor***

Example: Forum Nodes

- ***Uniquely identified*** by their name

  and the moderator node

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

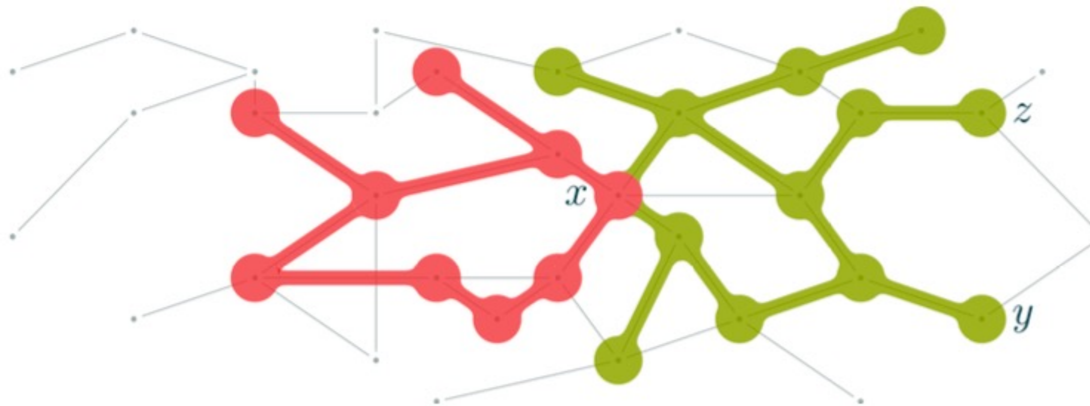
Scope is the set of all possible targets of a constraint

We set the scope to be all nodes labeled Forum

Descriptor determines key values for each target in the scope

We assign every node representing a forum a unique pair of name and person

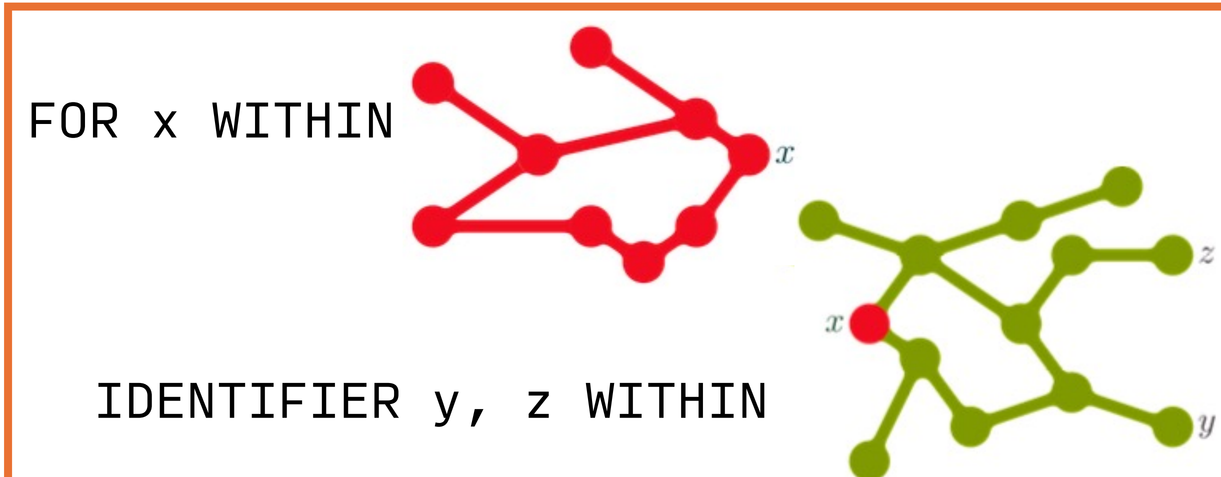
PG-Keys



Design requirements

- Flexible choice of key scope and descriptor of key values
- Keys for nodes, edges and properties
- Identify, reference and constrain objects
- Easy to validate

PG-Key Constraint



PG-Keys

Flexible Choice of Scope and Key Values

- Declaratively specify the scope and descriptor of the key
- PG-Keys is parameterized on a query language (e.g. GQL)

Example: Person Nodes

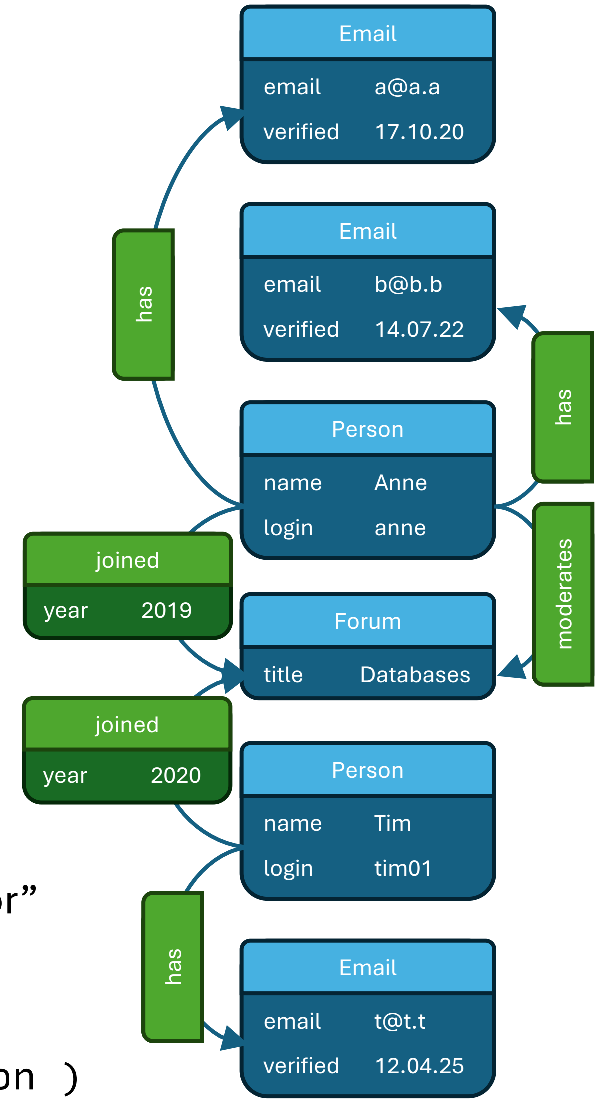
"each person is identified by their login"

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

Example: Forum Nodes

"each forum with a member is identified by its name and moderator"

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



PG-Keys

Keys for Nodes, Edges, and Properties

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

Example: Person Nodes

“each node labelled Person is identified by the value of property login”

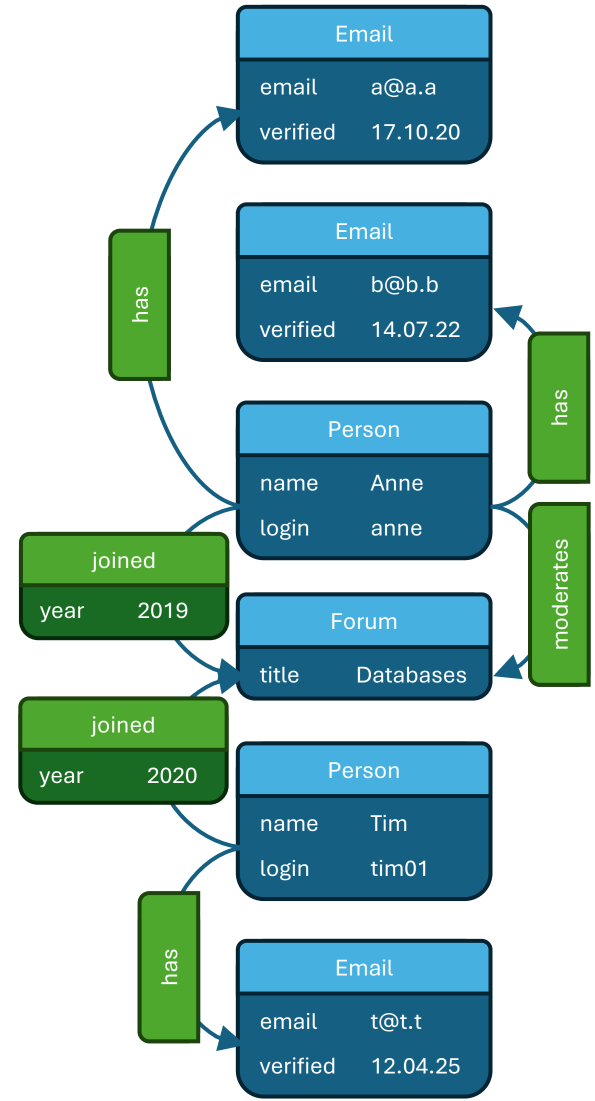
```
FOR p WITHIN (p: Person ) IDENTIFIER p. login
```

Example: Joined Relationships

“a person cannot join a forum twice,” i.e.,

“each edge labelled ‘joined’ is identified by its endpoints”

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



PG-Keys

Identify, Reference, and Constrain Objects

- Unique identification can be expressed with the qualifier `IDENTIFIER`

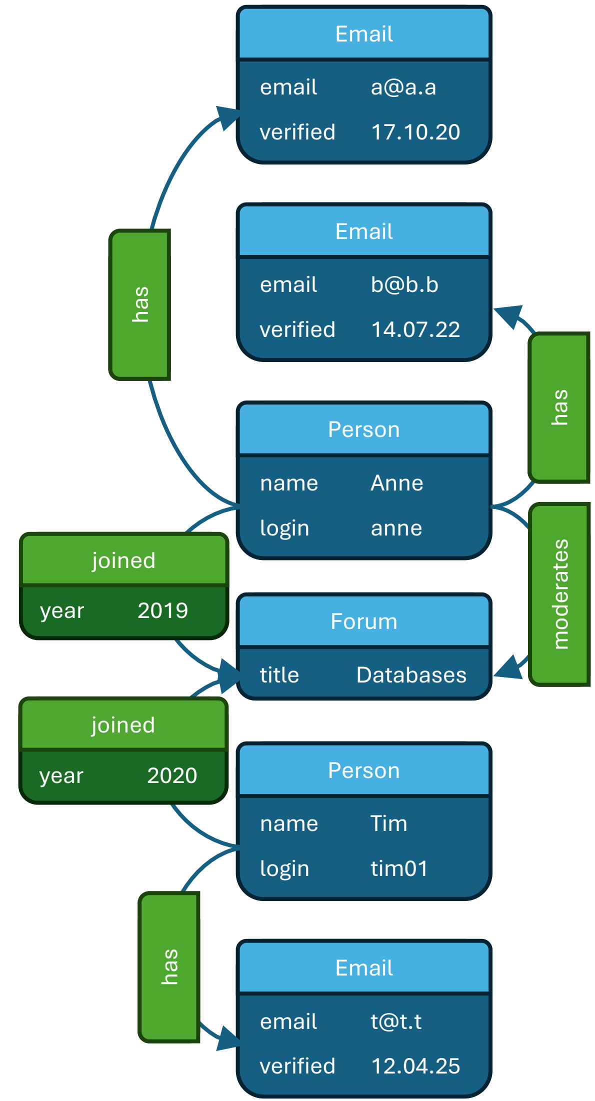
Example: Person Nodes

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

`IDENTIFIER` is the combination of the qualifiers

- EXCLUSIVE – *<u>no two targets</u>* in the scope can have the same key value
- MANDATORY – each target in the scope has *<u>at least one</u>* key value
- SINGLETON – each target in the scope has *<u>at most one</u>* key value

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



PG-Keys

Easy to Validate

- To check whether a constraint holds, we can run queries to find violations

Example

The key constraint

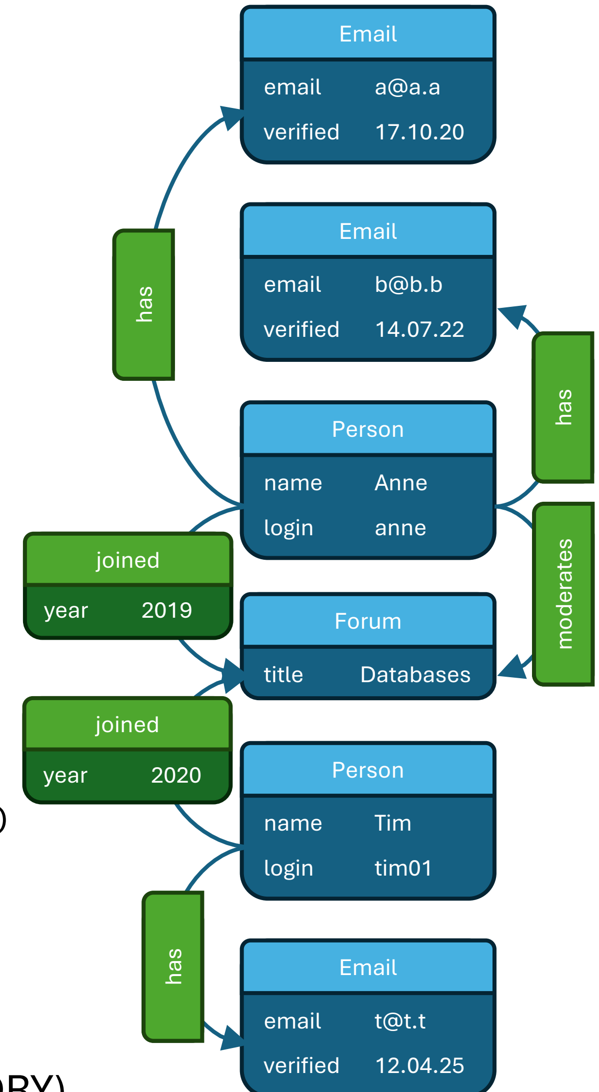
```
FOR p WITHIN (p: Person )
EXCLUSIVE MANDATORY e WITHIN (p)-[: has ]-(e: Email )
```

holds if both queries return *<u>no answers</u>*

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

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

(Alternatively, check for every p whether e is EXCLUSIVE MANDITORY)



Validating Property Graphs with a Schema

To summarize…

- a PG-Schema consists of:
  - PG-Types: node and edge types which can be found in the graph
  - PG-Keys: (key) constraints that need to hold in the graph, parameterized by a query language
- When the graph type is LOOSE: only the PG-Keys must hold
- Otherwise, it is STRICT:
  - Every node and edge in the graph must have at least one type

Limited Support for Schema in Systems

- Landscape for schema and constraints is diverse
  - Some systems offer property-based primary keys for nodes
  - Some support uniqueness
  - …

- There is no system that implements PG-Schema as described here
- Eventually, the vision is that all PG-Schema's features will be present in real systems

Constructing Graph Data

Clear for RDF, mostly uncharted territory for LPGs

Data Integration

- Integrating data from multiple heterogeneous sources is one of the main motivations for graph data
- Source data is almost never natively graph data
- How do we construct graph data?

→ Let's look at obtaining graph data from relational sources

Aside: Data Models & Formats

- **Data Models**: abstract, logical structures used to organize and manage data + a query language
  - Relational Model + SQL
  - Graph Models
    - RDF + SPARQL
    - LPG + GQL
- **Data Exchange/Serialization Formats**: (text-based) formats used to represent and transmit data
  - Relational data: no standard way, commonly:
    - CSV, Parquet, JSON, XML, SQL scripts
  - RDF data: different (standard) serializations:
    - Turtle, N-Triples, Jelly, …
  - LPGs: no standard way
    - GQL/Cypher scripts, GraphML

Side note:
Not everything fits neatly into this division. E.g., XML is a text-based format with a standard query language XQuery.

Typical Data Generation Workflow

1. Read data from a <u>source</u> data model (system or file)
2. Transform the source data into a new target structure
3. Output a file in a suitable data format for the <u>target</u> data model

→ This is a painful and error-prone process, two ways forward:

A. Use default translations between data models (inflexible)

B. Use a declarative description of the translations (more flexible)

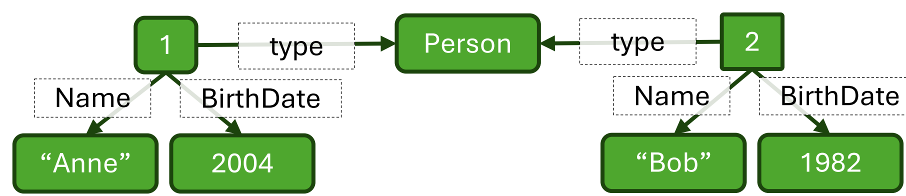
Transforming Data

For RDF, data exchange has always been at the forefront. Both (A) and (B) are standardized.

A. The RDF Direct Mapping defines a default translation from relational sources to RDF targets

Person

ID	Name	BirthDate
1	Anne	2004
2	Bob	1982



Transforming Data

For RDF, data exchange has always been at the forefront. Both (A) and (B) are standardized.

B. (R2)RML (Relational to RDF Mapping Language) defines a language for declarative rules for transforming relational or other sources to RDF targets

Abstractly, we define rules of the following form:

$$Q(s, p, o) \rightarrow (s, p, o)$$

with Q a query over the source that projects three variables corresponding to subject, predicate and object.

Transforming Data

Abstractly, we define rules of the following form:

$$Q(s, p, o) \rightarrow (s, p, o)$$

with Q a query over the source that projects three variables corresponding to subject, predicate and object.

- In the broader database exchange literature, these kinds of rules are known as Global-As-View mapping rules.
- When the source data is relational, Q is an SQL query → R2RML
- RML generalizes R2RML to sources that are not necessarily relational, but CSV, JSON, …

(R2)RML

Patient

National ID	Name	DateOfBirth	BloodType
1	A. Smith	1985	NULL
2	B. Green	1992	O+
3	C. Davis	1972	NULL
4	D. Johnson	1992	A+

We can read the semantics of a mapping rule as:

- If Q is just a table name, the mapping iterates of the rows of the table.
- Otherwise, in general, it iterates over the results of the query.

$$\text{Patient}(\text{NationalID}, \text{Name}) \rightarrow (\text{NationalID}, \text{name}, \text{Name})$$

```
<#TriplesMap1> [
  rr:logicalTable [ rr:tableName “Patient” ];
  rr:subjectMap [
    rr:template “http://patient.org/{NationalID}”;
    rr:class ex:Patient;
  ];
  rr:predicateObjectMap [
    rr:predicate ex:name;
    rr:objectMap [ rr:column “Name” ]; ] .
```

(R2)RML

Patient

National ID	Name	DateOfBirth	BloodType
1	A. Smith	1985	NULL
2	B. Green	1992	O+
3	C. Davis	1972	NULL
4	D. Johnson	1992	A+

We can read the semantics of a mapping rule as:

- If Q is just a table name, the mapping iterates of the rows of the table.
- Otherwise, in general, it iterates over the results of the query.

```
<#TriplesMap2> [
  rr:logicalTable [ rr:sqlQuery """
  SELECT NationalID, Name, DateOfBirth
  FROM Patient
  WHERE BloodType IS NOT NULL
  """
 ];
  rr:subjectMap [
    rr:template "http://patient.org/{NationalID}";
    rr:class ex:Patient;
  ];
  rr:predicateObjectMap [
    rr:predicate ex:name;
    rr:objectMap [ rr:column "Name" ];
  rr:predicateObjectMap [
    rr:predicate ex:dateOfBirth;
    rr:objectMap [ rr:column "DateOfBirth" ];
] .
```

RML Generalizes R2RML

- … but is not yet a W3C standard
- Allows for other data models and formats as sources
  - This means, the RML engine must support these formats
  - (It’s not magic)

```
{ "venue":
  { "latitude": "51.05",
    "longitude": "3.71” },
  "location":
  { "continent": " EU",
    "country": "BE",
    "city": "Brussels” }}
```

```
ex:Brussels a schema:City;
  pos:lat “51.05”;
  pos:long “3.71” .
```

```
<#VenueMapping> a rr:TriplesMap;
  rml:logicalSource [
    rml:source "Venue.json";
    rml:referenceFormulation ql:JSONPath;
    rml:iterator "$" ];
  rr:subjectMap [
    rr:template "http://ex.com/city/{location.city}";
    rr:class schema:City ];
  rr:predicateObjectMap [
    rr:predicate pos:lat;
    rr:objectMap [
      rml:reference "venue.latitude" ]];
  rr:predicateObjectMap [
    rr:predicate pos:long;
    rr:objectMap [ rml:reference "venue.longitude" ]].
```

source: rml.io/specs/rml/#example-JSON

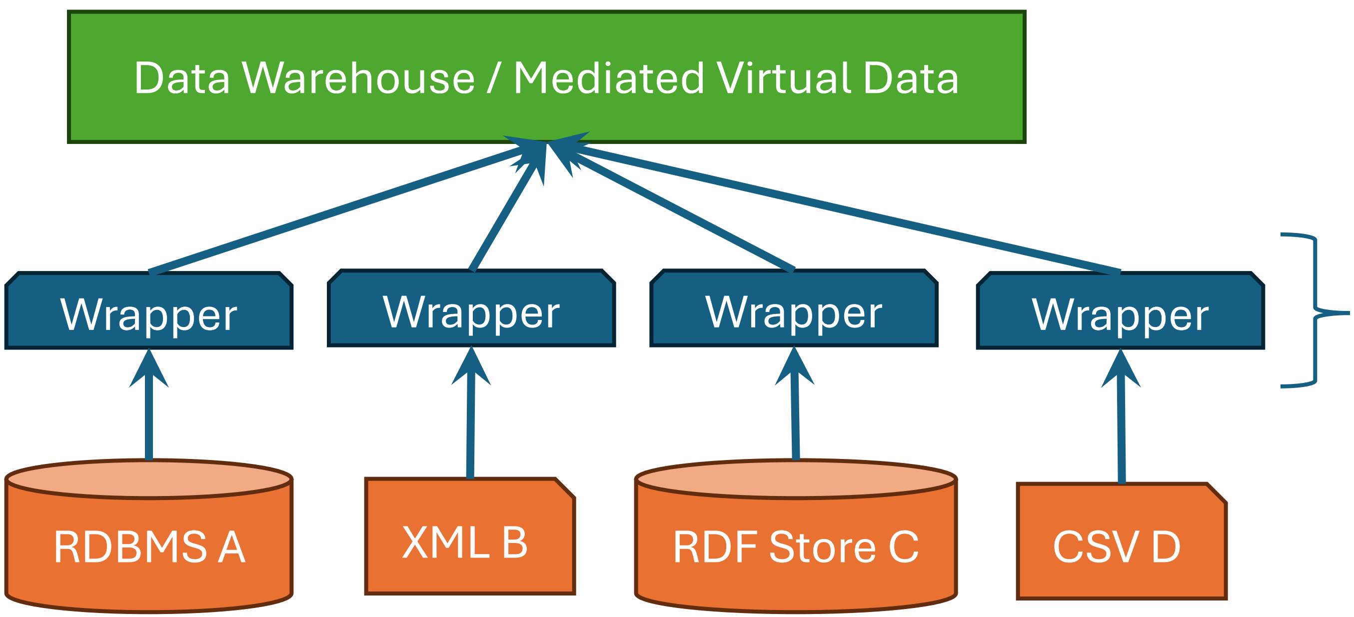
(Virtual) Data Warehousing

- **Data warehousing**
  - integrating data from multiple sources into one single (physical) source
  - you execute the mappings and materialize the data
- **Virtual data integration**
  - leave the data at the sources, only access at query time
  - you use the mappings to rewrite the query in order to execute them over the sources

- Warehousing is expensive when the sources are frequently updated → generally not up-to-date
- Virtual query answering is hard for heavy-duty *analytical* queries

(Virtual) Data Warehousing

- In the RDF world, known as Ontology Based Data Access (OBDA)
- Largely unexplored in the LPG world



- Send queries to sources and transform answers to target data model
- Declarative mapping rules (GAV-style)

Further Project Information

Sources of (Graph) Data

- Most open graph data is RDF:
  - Wikidata – https://www.wikidata.org/
  - DBPedia – https://dbpedia.org
  - YAGO – https://yago-knowledge.org/
  - Openstreetmap – https://wiki.openstreetmap.org/wiki/SPARQL_examples
  - DBLP – https://sparql.dblp.org
  - Pubchem – https://pubchem.ncbi.nlm.nih.gov/docs/rdf
  - UNIProt – https://sparql.uniprot.org

General purpose

Geo data

Computer Science scholarly graph

Life Sciences

- Non-graph interesting datasets:
  - https://data.europa.eu/
  - https://datasetsearch.research.google.com
  - Use other public APIs to create a graph dataset
  - Webscraping …
- Using public APIs/query services is possible, or use your own

Example Project Directions

- EU Policy and Funding Tracer
  - Build a graph that connects EU-funded projects to the organizations that received the funds and the public policies they relate to, enabling analysis of funding impact.
  - **Data sources**: data.europa.eu, DBPedia, Wikidata …
- Smart City Logistics Planner
  - Model a city's road network, public transport routes, and key locations as a graph to find the most efficient delivery or travel routes using multiple modes of transport.
  - **Data sources**: OpenStreetMap, public transit authority APIs (e.g., Wiener Linien), Wikidata for points of interest, …
- Graph Transformation and Benchmarking (advanced idea → expand to master thesis)
  - A toolkit that takes a non-graph dataset (e.g., CSV), constructs both RDF and Property Graph data, loads them into different corresponding databases, and runs a series of equivalent queries to benchmark their performance.
- **We highly recommend thinking about your project from the start**, and use the lectures to get a more concrete idea.
- These are not complete descriptions, use the rubric to complete the project idea and talk to us if in doubt

Domain Specific

Research Driven

Graph Data-Driven Applications

- Obvious example: Wikidata
  - Uses the schema-less approach to accommodate all kinds of knowledge
  - Users can add/edit/remove “edges” “triples” from the data
  - Supports querying with SPARQL
- Rijksmuseum Amsterdam (RDF and LPGs!):
  - Powered by graph data: https://www.rijksmuseum.nl/en/collection
  - Data is available: https://data.rijksmuseum.nl/docs/
  - (similarly, https://www.britishmuseum.org/collection)
- Europeana: promotes Europe’s digital cultural heritage
  - Powered by graph data: https://www.europeana.eu/en
  - Data is available: https://pro.europeana.eu/en/

Project Direction: creating new graph data

- Constructing your own graph database around a certain topic
  - Extending an existing dataset
  - Translating an existing dataset into graphs
  - Combining multiple datasets + construct parts of graphs using publicly available API's
  - Recording provenance of the data changes/ data construction process
- The application itself is not the focus, but it should demonstrate the usefulness of the graph data that you created
  - Demonstrates the usefulness of your specific constructed graph

Further Inspiration: Data Online

- Musea, cultural heritage (see previous links)
- International Consortium of Investigative Journalists
  - https://offshoreleaks.icij.org (LPG)
- Open Data Vienna
  - Mobility data
- Open Street Map (also with RDF: Linked Geo Data)
- Google's Data Commons
  - Discover data about economics, demographics, health, climate, ...
- Life Sciences...

Graph Database Technologies

- RDF
  - Comunica (open-source, decentralized web querying)
  - GraphDB Free (free-to-use)
  - Jena TBD2 (open-source, community developed)
  - MilleniumDB (open-source, in development, also LPG)
  - Ontop (OBDA, virtual RDF graph over relational data)
  - QLever (open-source, in development, no good support for updates)
  - RDF4J (open-source, community developed)
  - Virtuoso Open Source (open-source, commercial version available)
  - Usage of smaller, in-memory, engines are discouraged (e.g., RDFLib)
- LPG
  - MemGraph (like Neo4J, but faster according to themselves)
  - MilleniumDB (open-source, in development, also RDF)
  - Neo4J (recommended)
- You are free to choose any of these, or others outside of this list

Further Pointers: Working with Neo4J

- Written in Java, you can use a lot of custom functions
  - Most popular is the "apoc" library, which you can call in Neo4J
  - You can easily write your own user defined procedures as a plugin
- There is also the NeoSemantics project within Neo4J which provides the functionality of working with RDF in their system

Further Pointers: Great Relational DBMSs

- DuckDB is an open-source in-process SQL OLAP DBMS (paper)
    - High-tech state-of-the art relational data management
    - Ideal for processing data in-memory
    - It even has SQL/PGQ!
- For you project:
    - If you work with table-like data, e.g., CSV or information retrieved directly from APIs, you can load them into DuckDB to further process it
    - For example, in workflows with R2RML, you can load your CSV files easily in DuckDB and then use the mapping rules.
- Umbra DB is another open-source high-tech DBMS (paper)