Universität Zürich[UZH]

Institute of Political Science
www.ipz.uzh.ch

From Entities to Edges

A Practical Introduction to Knowledge Graphs

Laurence Brandenberger, Yaren Durgun

Introductions

- Laurence Brandenberger
- PhD (Political science) in 2019 from the University of Bern
- Postdoc and senior assistance at ETH Zurich
  - at an interdiciplinary chair
  - focusing on complex systems
- Now I'm at the Department of Politics at Uni Zurich,
- Topics of interest:
  - political methodology
  - political elite and networks
  - legislative studies

Today's Workshop Outline

09:30–09:45 — Welcome

- Introduction and goals of the workshop
- Overview of materials and setup

09:45–10:15 — Introduction to Knowledge Graphs

- What are Knowledge Graphs?
- Why should social scientists care?
- Core concepts: nodes, relations, properties

10:15–10:45 — Benefits & Use Cases

- Integrating and cleaning messy data
- Transparency and reproducibility
- Long-term data maintenance

10:45–11:00 — Case Study

- The DemocraSci Knowledge Graph (1891–today)
- Lessons learned from the Swiss Parliament data

11:00–11:30 — Break

- Coffee, discussion, setup for hands-on session

11:30–12:30 — Hands-on in R

- Build a Neo4j Knowledge Graph from scratch
- Validate and query your graph
- Explore example datasets (Northwind & Parliament)

13:00+ — Closing Discussion

- Open Q&A and exchange of ideas
- Where to go next: data2neo, DemocraSci, and beyond

What You'll Learn Today

- **Understand** what a Knowledge Graph is
  - Core parts: nodes, relationships, and properties
  - How graphs differ from traditional relational databases
- **Recognize** why KSs matter for social science
  - Integrating diverse data sources
  - Making research more transparent and reusable
- **Build and validate** a small Knowledge Graph yourself
  - From CSV or Excel data → Neo4j database
  - Using `data2neo` and `Cypher` queries
- **Apply** what you learn to real-world cases
  - The DemocraSci KG on the Swiss Parliament
  - The UK Parliament dataset as a hands-on exercise

What ChatGPT5.o thinks our workshop is about

Part A

Introduction to Knowledge Graphs for Social Scientists

or

Why would you care about knowledge graphs?

What is a Knowledge Graph?

Definition by IBM

"A knowledge graph, also known as a semantic network, represents a network of real-world entities (such as objects, events, situations or concepts) and illustrates the relationship between them. This information is usually stored in a graph database and visualized as a graph structure, prompting the term knowledge "graph.""

Definition by the Alan Turing Institute

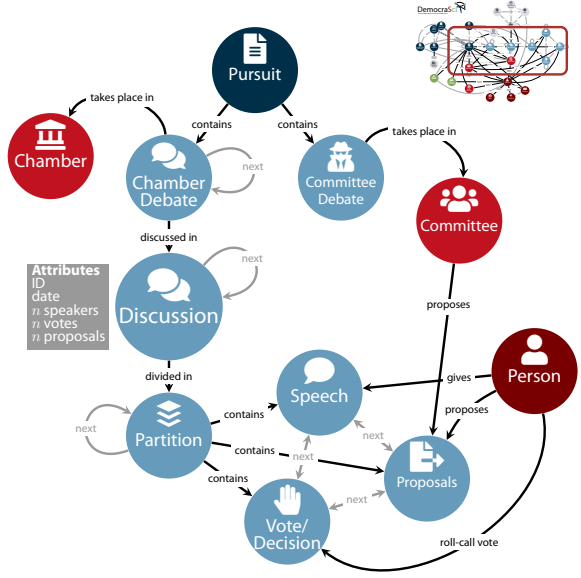
"Knowledge graphs (KGs) organise data from multiple sources, capture information about entities of interest in a given domain or task (like people, places or events), and forge connections between them."

Definition by neo4j

"A knowledge graph is a design pattern for storing, organizing, and accessing interrelated data entities, including their semantic relationships."

What is a Knowledge Graph? → for Social Sciences

- A **knowledge graph (KG)** is a type of **database** designed to represent relationships.
- Unlike a spreadsheet or SQL table:
  - It has **no fixed rows or columns**.
  - Every **entity (node)** and **relationship (edge)** is stored explicitly.
- KGs describe both:
  - **Entities and their attributes** (e.g., *MP, party, committee*)
  - **Relations between entities** (e.g., *MP* → *sponsors* → *Bill*, *MP* → *memberOf* → *Party*)
- This makes them ideal for **complex, relational social science data**.



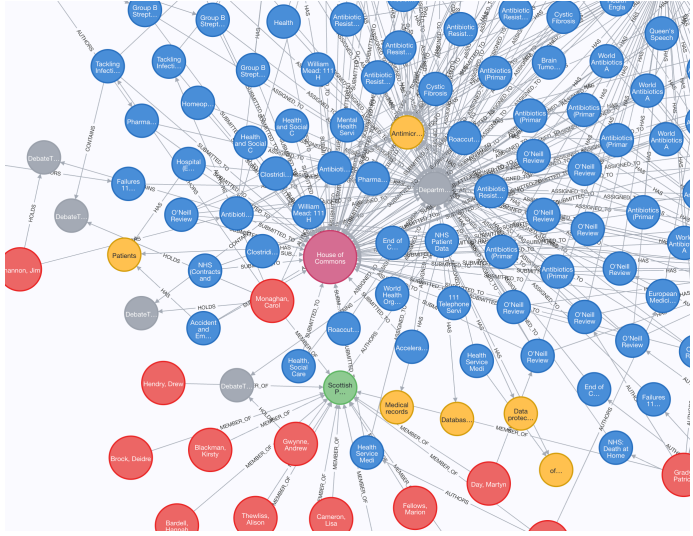
An Example: UK parliament data

- Let's say you have 1 excel file like this:

Type	Ref	Date	Title	Member	Member Party	Lead Member	Lead Member Party	Answering Member	Answering Member Party
Written questions	105387	2022-07-12 00:00:00	Antimicrobials: Drug Resistance	Shannon, Jim	Democratic Unionist Party	Shannon, Jim	Democratic Unionist Party		
Members' contributions	724 c333	2022-07-12 00:00:00	Engagements	Sunak, Rishi	Conservative Party				
Written questions	98674	2022-07-12 00:00:00	Livestock: Drug Resistance	Sheerman, Barry; Spencer, Mark	Labour Party; Cooperative Party; Conservative Party	Sheerman, Barry	Labour Party; Cooperative Party	Spencer, Mark	Conservative Party
Written questions	104350	2022-06-12 00:00:00	Antimicrobials: Drug Resistance	Clark, Feryal	Labour Party	Clark, Feryal	Labour Party		

An Example: UK parliament data

- You can transform it into a KG
- each cell becomes a node or a relation



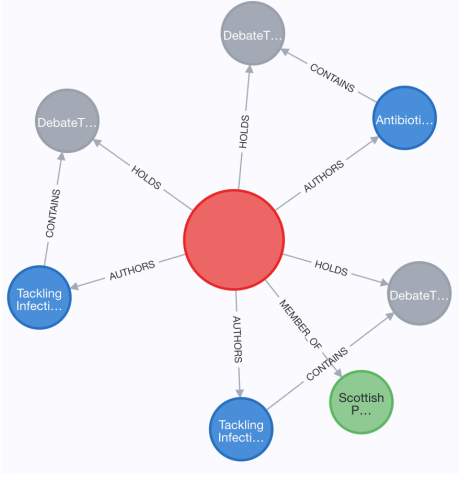
Basic Foundation of Knowledge Graphs: Nodes and Relations

Nodes

- Represent **entities** — the "things" in your data.
- Each node has a **label** (type) and a set of **properties**.
- Examples:
  - *MP, Party, Bill, Committee*
  - *Country, Year, Institution*
- Think of nodes as the "nouns" in your research domain.

Relations

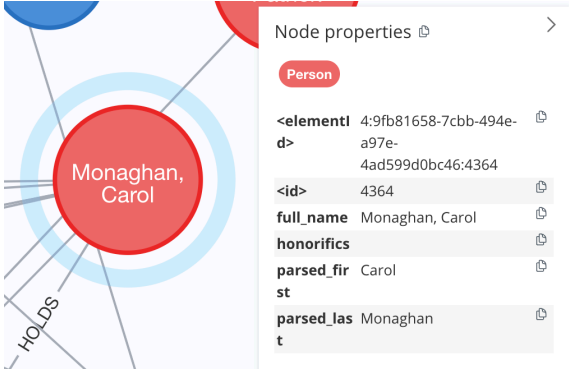
- Represent **connections or interactions** between nodes.
- Have a **direction** and often a **type**.
- Examples:
  - *MP* → *memberOf* → *Party*
  - *MP* → *sponsors* → *Bill*
  - *Committee* → *discusses* → *Bill*
- Think of relations as the "verbs" linking entities.



Each node (circle) is connected by a typed relation (arrow).

Properties on Nodes

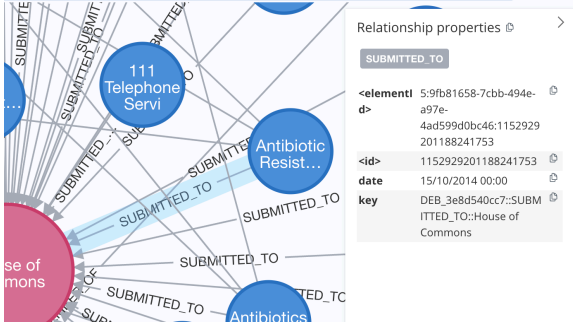
- Each **node** can have one or more **properties** describing its characteristics.
- Properties are stored as key–value pairs.
- They make your entities **informative and searchable**.
- Examples:
  - MP node: `name = "Anna Keller"`, `gender = "female"`, `birthYear = 1976`
  - Party node: `name = "Social Democratic Party"`, `abbrev = "SP"`
  - Bill node: `id = 23/2021`, `title = "Climate Protection Act"`
- Properties help you query or filter nodes easily:
  - `MATCH (m:MP) WHERE m.gender = "female" RETURN m`



Each node (entity) carries attributes as key–value pairs.

Properties on Relations

- Relations can also have **properties** describing the **nature or context** of the connection.
- Like node properties, these are stored as key–value pairs.
- Useful when relationships include **temporal**, **quantitative**, or **categorical** details.
- Examples:
  - *(:MP)-[r:SPONSORS]->(:Bill)*
    `r.date = "2023-05-17"`
  - *(:MP)-[r:COLLABORATES_WITH]->(:MP)*
    `r.strength = 0.82`
  - *(:Country)-[r:TRADES_WITH]->(:Country)*
    `r.value = 1200000`, `r.year = 1913`
- Relation properties allow for more **expressive and analytical queries**.



Relations can carry properties such as date, type, or intensity.

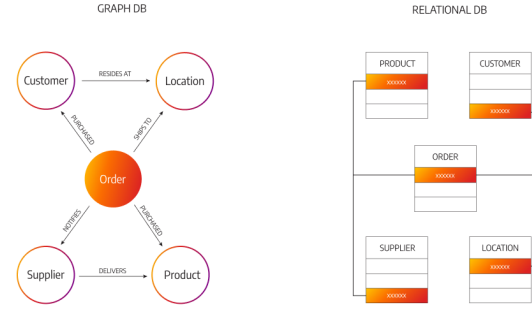
How Knowledge Graphs Differ from Relational Databases

Relational Databases (SQL, Excel)

- Data organized in **tables** (rows and columns).
- Relationships are defined **indirectly** via keys and joins.
- Schema is usually **fixed/rigid**.
- Well suited for **structured, tabular data**.

Knowledge Graphs (Neo4j)

- Data stored as **nodes and relationships**.
- Connections are **explicit** and **natively stored**.
- Schema is **flexible**
- Ideal for **complex, interconnected data**.



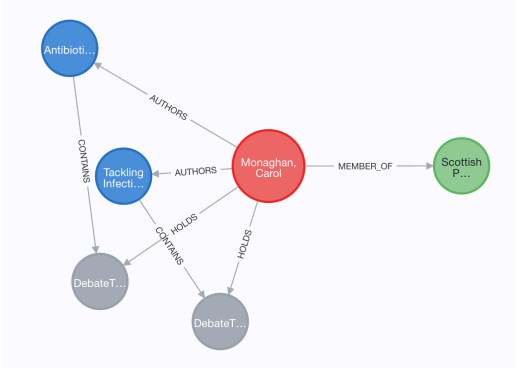
Check out `https://www.techtarget.com/searchdatamanagement/feature/Graph-database-vs-relational-database-Key-diffe`

Thinking in Triples: Subject – Relation – Object

- Every statement in a knowledge graph can be expressed as a **triple**:

$$\text{(Subject)} \xrightarrow{\text{Relation}} \text{(Object)}$$

- Each triple captures a single **fact** about the world.
- Examples:
  - `(Anna Keller) -- memberOf` → `(Social Democratic Party)`
  - `(Bill 23/2021) -- sponsoredBy` → `(Anna Keller)`
  - `(Committee on Health) -- discusses` → `(Bill 23/2021)`
- The full knowledge graph is simply the **union of all triples**.
- Triples can easily be **visualized**, **queried**, and **expanded**.



Example: MP – memberOf → Party

Part B

Benefits & Limitations

or

When is it useful, when not?

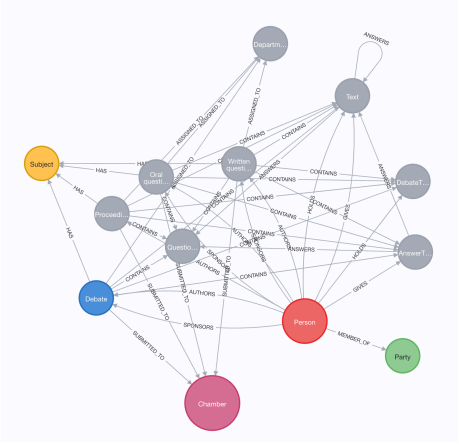
When You *Don't* Need a Knowledge Graph

- You only have **one dataset or spreadsheet**.
- Your data are **flat** — no links to other sources.
- You only need the data for a **single project or paper**.
- Relationships between entities are **simple or irrelevant**.
- Your workflow is mostly:
  - *load data* → *clean* → *analyze* → *publish*.
- In short: a knowledge graph is **overkill** if your data do not live beyond one study.

mp	bill	role	co_sponsor
74	15.4089	no (co)sponsor	0
91	15.4089	no (co)sponsor	0
173	15.4089	no (co)sponsor	0
214	15.4089	no (co)sponsor	0
224	15.4089	no (co)sponsor	0
307	15.4089	no (co)sponsor	0
354	15.4089	no (co)sponsor	0
487	15.4089	no (co)sponsor	0
502	15.4089	no (co)sponsor	0
514	15.4089	no (co)sponsor	0
519	15.4089	no (co)sponsor	0
724	15.4089	no (co)sponsor	0
806	15.4089	no (co)sponsor	0
1071	15.4089	no (co)sponsor	0
1104	15.4089	no (co)sponsor	0
1108	15.4089	no (co)sponsor	0
1109	15.4089	no (co)sponsor	0
1111	15.4089	no (co)sponsor	0

When You *Do* Need a Knowledge Graph

- You want to **integrate diverse datasets or formats**.
  - E.g., combine `CSV` files, `JSON` data, or multiple SQL databases.
  - Link entities that appear across sources (e.g., MPs, committees, bills).
- You need to **iteratively expand or update** your data.
  - You continuously collect or annotate new data.
  - You want to avoid breaking the structure each time.
- You care about **transparency and reproducibility**.
  - Relationships and provenance are explicit.
  - Each node and relation can be traced and queried.
- You want to **validate data quality** through *constraints*.
  - Example: every `MP` must belong to exactly one `Party`.



Part C

Case Study: The DemocraSci Knowledge Graph

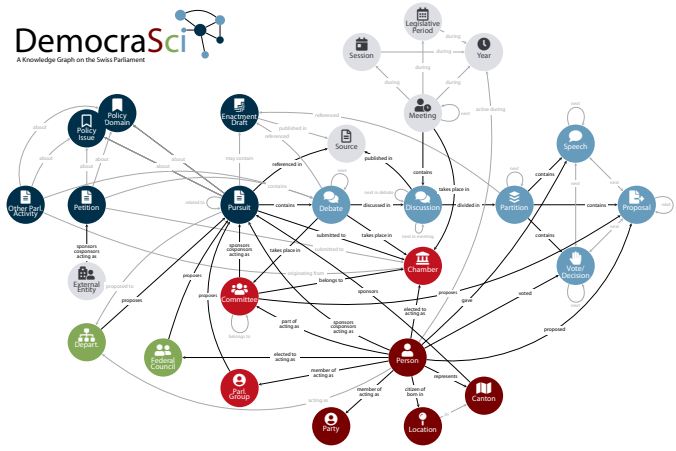
or

How we built a KG of the Swiss parliament, spanning data from over 130 years

The DemocraSci Knowledge Graph

Relational Graph Database:

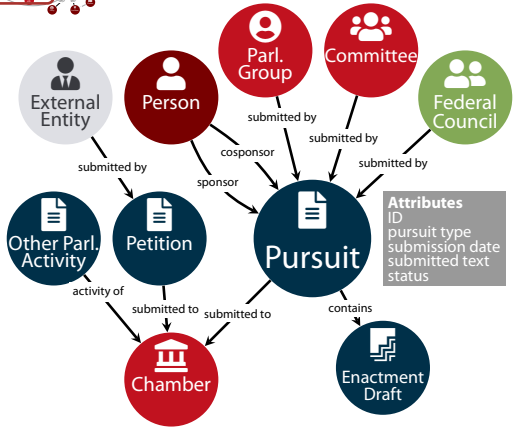
- **Complete record of parl. activities**
- **1891 - today**
- ..links pursuits to MPs
- ..links MPs to debates
- ..links pursuits to debates
- Our KG is based on **parliamentary pursuits**



Brandenberger, L., Schlosser, S., Salamanca, L., Gasser, L., Balode, M., Minder, J., Jung, V., Durgun, Y., Babic, L., Perez-Cruz, F., and Schweitzer, F. (2025). Democrasci: A knowledge graph on the swiss parliament. *Paper submitted to Nature. Data Descriptor.*

Focus on Pursuits

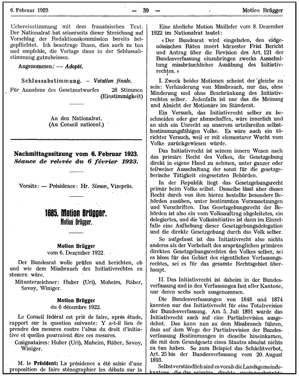
- Our database is centered around pursuits
- We track
  - .. sponsorship + date
  - .. cosponsorship
  - .. the content
  - .. how the pursuit moves through the chambers
- in total we have over **95k pursuits** (1891-today)
- Recently: about 9k per legislative period



Document sources

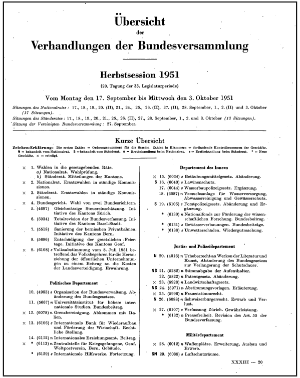
Amtliches Bulletin

- Records of propositions and discussions
- 208,788 pages
- 35,789 documents



Summary of proceedings

- Records of all parliamentary bills
- 19,081 pages
- 426 documents

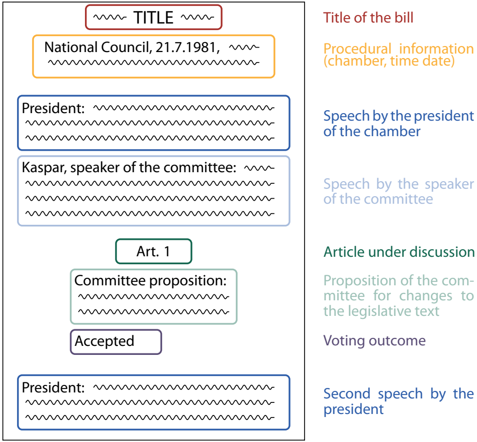


Online Database

- Online records of the parliament
- From 1995 onwards
- Records are overwritten (no history)

What we Extracted from the Bulletin PDFs

Stylized Page



Example Page

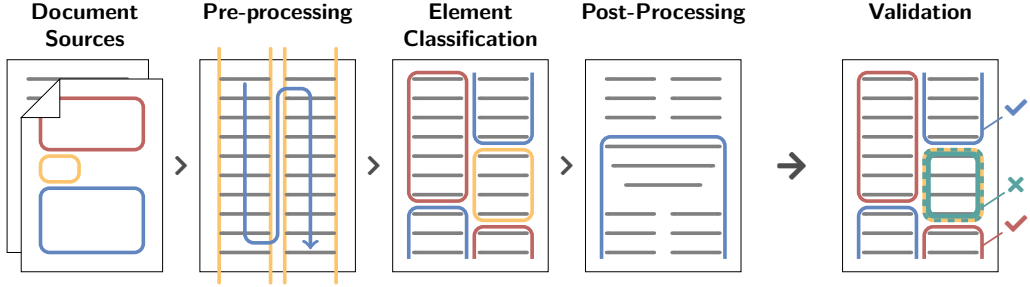
Records of:

- Parliamentary **speeches**
- Legislative **propositions**
  - Bill texts
  - Law articles and proposed changes
- **Votes**
  - Decisions
  - Roll-call votes

- Every page looks different

Processing Pipeline

- We came up with a processing pipeline



Step 1: determine which elements to extract.

Step 2: Prepare xml documents by determining margins and adjusting text flow.

Step 3: Classification of text elements with the help of a supervised model.

Step 4: Improve element classification by separating documents and merging page breaks.

Step 5: Validate all aspects of the extraction process through manual assessment.

- Step 3 entails classification → pdf2data

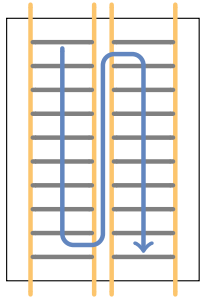
Salamanca, L., Brandenberger, L., Gasser, L., Schlosser, S., Balode, M., Jung, V., Perez-Cruz, F., and Schweitzer, F. (2024). Processing large-scale archival records: The case of the swiss parliamentary records. *Swiss Political Science Review*, 30(2):140–153

Segmenting Documntes with `pdf2data`

- **segment** PDF documents
- **help annotate** different parts of the document
- **train a model** to automatically identify elements on pages
- **extract** the text
- ..and **save** it in a database (graph, csv, json)

`pdf2data` takes on these two steps:

Pre-processing



Step 2: Prepare xml documents by determining margins and adjusting text flow.

Element Classification

Step 3: Classification of text elements with the help of a supervised model.

Salamanca, L., Meyer, M., Brandenberger, L., Schlosser, S., Campos-Schweitzer, J., Schweitzer, F., and Perez-Cruz, F. (2023). pdf2data: A tool for extracting information from pdf documents

When To Use `pdf2data`

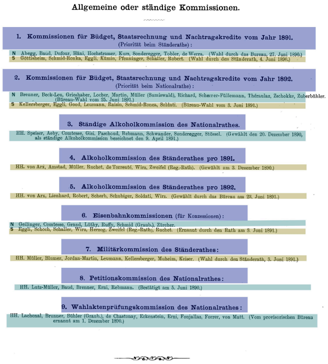
`pdf2data` needs **repeated elements** on text documents

1) Identify page content

- identify full pages
- large-scale projects

2) Classify text boxes

- annotate text boxes
- large/small-scale projects



Allgemeine oder ständige Kommissionen.

1. Kommissionen für Büdget, Staatsrechnung und Nachtragskredite vom Jahr 1891.
(Priorität beim Ständerathe):
N Abegg, Baud, Dufour, Häni, Hochstrasser, Kurz, Sonderegger, Tobler, de Werra. (Wahl durch das Bureau, 27. Juni 1890.)
S Göttisheim, Schmid-Ronka, Eggli, Kämin, Pfenninger, Schaller, Robert. (Wahl durch den Ständerath, 4. Juni 1890.)

2. Kommissionen für Büdget, Staatsrechnung und Nachtragskredite vom Jahr 1892.
(Priorität beim Nationalrathe):
N Brenner, Beck-Leu, Grieshaber, Locher, Martin, Müller (Sumiswald), Richard, Scherrer-Füllemann, Théraulaz, Zschokke, Zuberbühler. (Büreau-Wahl vom 25. Juni 1891.)
S Kellersberger, Eggli, Good, Leumann, Raisin, Schmid-Ronca, Soldati. (Büreau-Wahl vom 3. Juni 1891.)

3. Ständige Alkoholkommission des Nationalrathes.
HH. Speiser, Aeby, Comtesse, Gisi, Paschoud, Rebmann, Schwander, Sonderegger, Stössel. (Gewählt den 20. Dezember 1890, als ständige Alkoholkommission bezeichnet den 9. April 1891.)

4. Alkoholkommission des Ständerathes pro 1891.
HH. von Arx, Amstad, Müller, Ruchet, de Torrenté, Wirz, Zweifel (Reg.-Rath). (Gewählt am 3. Dezember 1890.)

5. Alkoholkommission des Ständerathes pro 1892.
HH. von Arx, Lienhard, Robert, Scherb, Schubiger, Soldati, Wirz. (Gewählt durch das Büreau am 23. Juni 1891.)

6. Eisenbahnkommissionen (für Konzessionen):
N Geilinger, Comtesse, Grand, Lüthy, Ruffy, Schmid (Graub.), Zürcher.
S Eggli, Schoch, Schaller, Wirz, Herzog, Zweifel (Reg.-Rath), Ruchet. (Ernannt durch den Rath am 3. Juni 1891.)

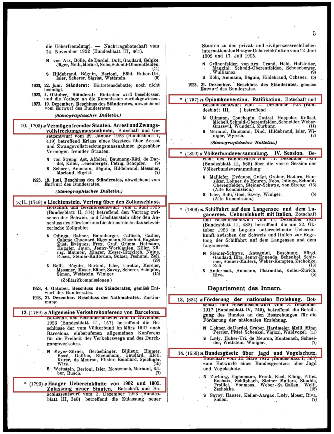
7. Militärkommission des Ständerathes:
HH. Müller, Blumer, Jordan-Martin, Leumann, Kellersberger, Muheim, Keiser. (Wahl durch den Ständerath, 3. Juni 1891.)

8. Petitionskommission des Nationalrathes:
HH. Lutz-Müller, Baud, Brenner, Erni, Rebmann. (Bestätigt am 5. Juni 1890.)

9. Wahlaktenprüfungskommission des Nationalrathes:
HH. Lachenal, Brunner, Bühler (Graub.), de Chastonay, Eckenstein, Erni, Fonjallaz, Forrer, von Matt. (Vom provisorischen Büreau ernannt am 1. Dezember 1890.)

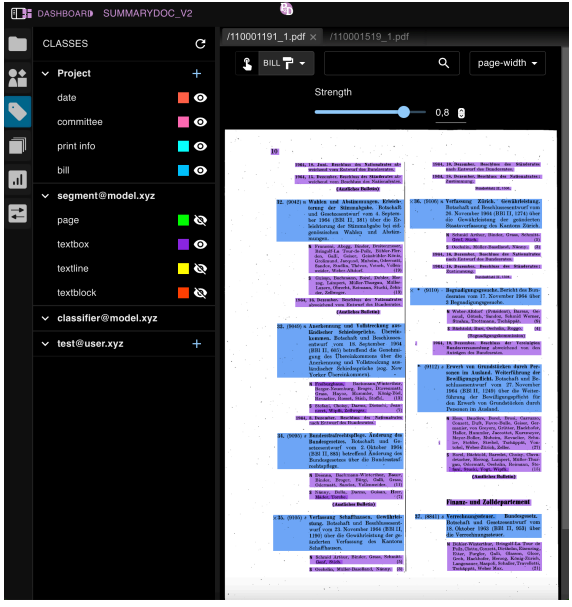
1) Classify text lines

- annotate single text lines
- large/small-scale projects



How pdf2data works

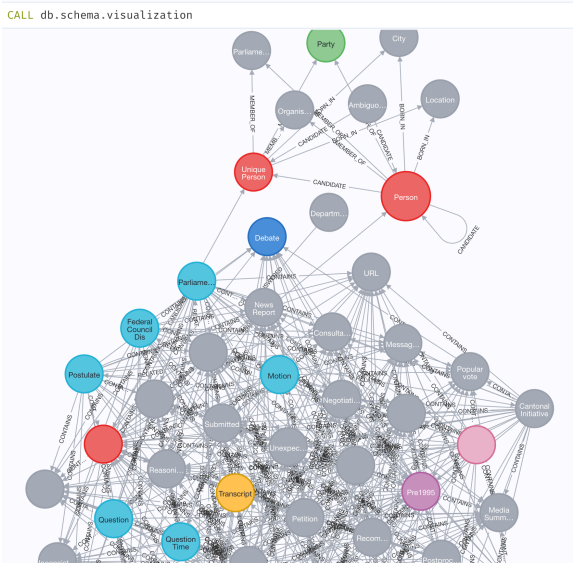
1. import documents
2. automatically let documents be **pre-processed**
3. **annotate** a few documents
4. **train a model** to detect and separate the elements
   - small batch: help with annotations
   - large batch: predict on unseen documents
5. **extract elements** and export



Querying the DemocraSci Knowledge Graph: Schema

```
CALL db.schema.visualization();
```

- Only works well on **small graphs**!
- Limited to **500 relationships** in Neo4j Browser.
- For full schema: `CALL db.schema.nodeTypeProperties();`

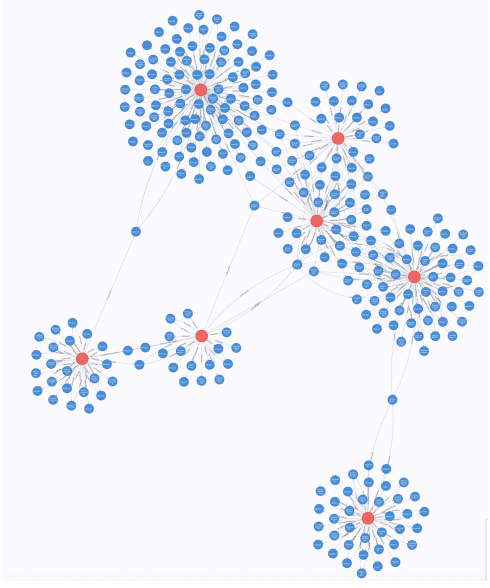


Schema visualization in Neo4j Desktop (truncated at 500 rels).

Querying the DemocraSci Knowledge Graph: Simple query

```
MATCH (mp:Person)-[r:SPONSORS]->(p:Pursuit)
RETURN mp, p
```

- `MATCH` is the most fundamental Cypher command — it finds patterns in your graph.
- You can assign **short aliases** to nodes (e.g., `mp` for Member of Parliament).
- `-[r:SPONSORS]->` specifies a relationship type and its direction.
- `RETURN` defines what to display — here: all MPs and the pursuits they sponsor.
- Neo4j Browser will visualize the results as a network of nodes and edges.



Querying the DemocraSci Knowledge Graph: Returning data tables

```
MATCH (mp:Person)-[r:SPONSORS]->(p:Pursuit)
RETURN mp.last_name AS Sponsor_LastName, mp.first_name AS Sponsor_FirstName, p.title_de AS
    PursuitTitle
LIMIT 20;
```

```
MATCH (mp:Person)-[r:SPONSORS]->(p:Pursuit)
RETURN mp.last_name AS Sponsor_LastName, mp.first_name AS Sponsor_FirstName, p.title_de AS PursuitTitle
LIMIT 20;
```

	Sponsor_LastName	Sponsor_FirstName	PursuitTitle
1	"Bischof"	"Hardi"	"Postulat Bischof Abgastest für Motorfahrzeuge der Grenzgänger"
2	"Bischof"	"Hardi"	"Einfache Anfrage Bischof Video- und Computerspiele"
3	"Bischof"	"Hardi"	"Einfache Anfrage Bischof Zentralstelle gegen die organisierte Kriminalität"
4	"Bischof"	"Hardi"	"Einfache Anfrage Bischof Mängel an Einrichtungen der IV"
5	"Bischof"	"Hardi"	"Einfache Anfrage Bischof Tod der Meeresschildkröten in Indonesien"
6	"Bischof"	"Hardi"	"Einfache Anfrage Bischof Gefahrgüter"

Querying the DemocraSci Knowledge Graph: Counting entities

```
MATCH (p:Party)<-[:MEMBER_OF]-(mp:Person)-[:SPONSORS]->(b:Bill)
RETURN p.name AS Party, COUNT(DISTINCT b) AS NumBills
ORDER BY NumBills DESC;
```

Party	NumBills
"Sozialdemokratische Partei der Schweiz"	17206
"Schweizerische Volkspartei"	12402
"FDP.Die Liberalen"	9354
"Christlichdemokratische Volkspartei der Schweiz"	9288
"Grüne Partei"	6444
"Die Mitte"	2598
"Grünliberale Partei"	1253
"Partei der Arbeit der Schweiz"	1132

Part D

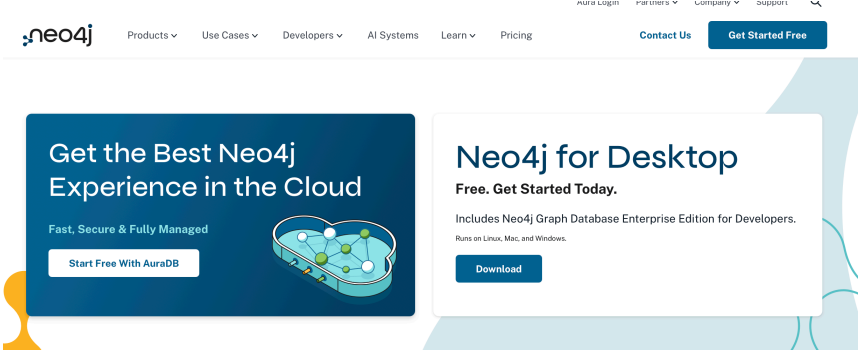
Hands-On: Creating a Neo4j KG using data2neo

or

From a simple example to a more realistic one

Download Neo4j

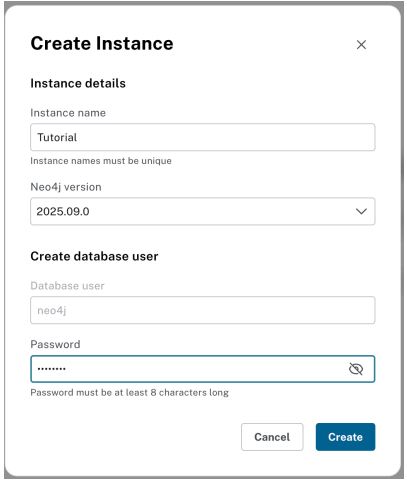
- Download Neo4j Desktop from **neo4j.com/download**.
- You will need to create a (free) Neo4j account.
- **Personal note:** Don't worry — Neo4j Desktop is free. They might prompt you to upgrade, but you can safely ignore it. Even large graphs (like the DemocraSci KG) run perfectly on the free tier.
- Once installed, launch **Neo4j Desktop** to create your first local database.



Neo4j Desktop download page.

Neo4j Desktop: Create a Database Instance (v2.x)

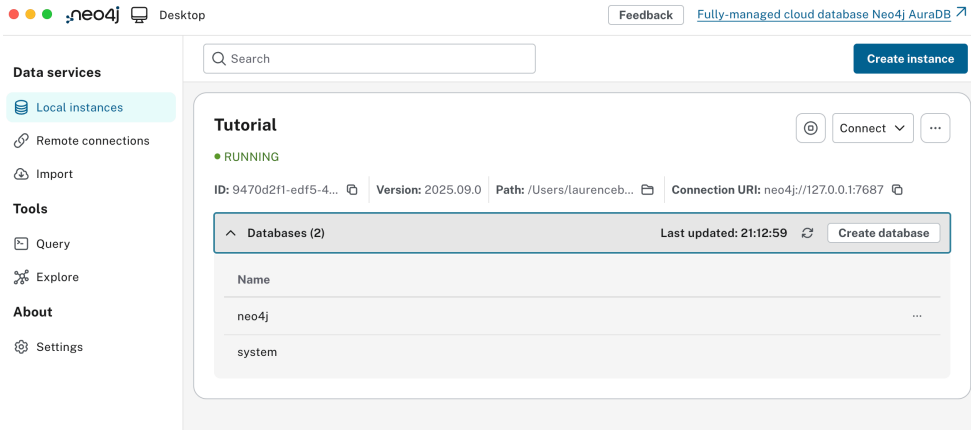
- Open **Neo4j Desktop**.
- Click on **Create instance** in the main window.
- Choose a descriptive name, e.g. `Northwind_KG`.
- Set a password — you will need it later to connect from R or Python.
- Optionally adjust the Neo4j version (default is fine).
- Click **Create**
- Wait until the status shows **Running** — your local Neo4j server is ready.



Creating a local database instance in Neo4j Desktop 2

Neo4jDesktop: Instance Setup

- You always have a database loaded (neo4j) = basis
- Here, we'll load in our data

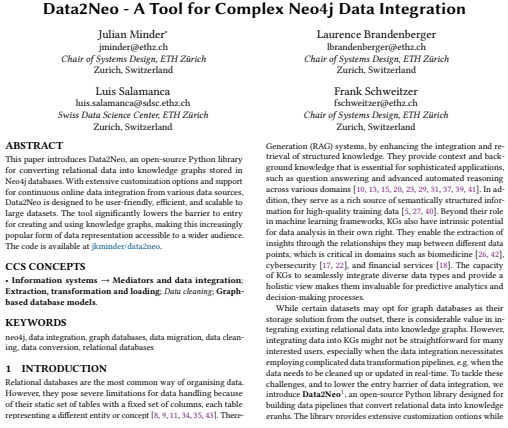


Tool to Convert Data to Neo4j: `data2neo`

- **data2neo** is a Python package developed by Julian Minder — it helps convert structured data into a Neo4j knowledge graph.
- Once your local Neo4j instance is **running**, switch to Python.
- Install the package in your terminal:

```
pip install data2neo
```

- It provides a simple schema-based way to map tables or DataFrames to nodes, properties, and relationships.
- Ideal for automating imports from CSV, Excel, or SQL databases into your graph.



Data2Neo - A Tool for Complex Neo4j Data Integration

Julian Minder*
jminder@ethz.ch
Chair of Systems Design, ETH Zürich
Zurich, Switzerland

Laurence Brandenberger
lbrandenberger@ethz.ch
Chair of Systems Design, ETH Zürich
Zurich, Switzerland

Luis Salamanca
luis.salamanca@sdsc.ethz.ch
Swiss Data Science Center, ETH Zürich
Zurich, Switzerland

Frank Schweitzer
fschweitzer@ethz.ch
Chair of Systems Design, ETH Zürich
Zurich, Switzerland

ABSTRACT

This paper introduces Data2Neo, an open-source Python library for converting relational data into knowledge graphs stored in Neo4j databases. With extensive customization options and support for continuous online data integration from various data sources, Data2Neo is designed to be user-friendly, efficient, and scalable to large datasets. The tool significantly lowers the barrier to entry for creating and using knowledge graphs, making this increasingly popular form of data representation accessible to a wider audience. The code is available at jkminder/data2neo.

CCS CONCEPTS

• **Information systems → Mediators and data integration**; **Extraction, transformation and loading**; *Data cleaning*; **Graph-based database models**.

KEYWORDS

neo4j, data integration, graph databases, data migration, data cleaning, data conversion, relational databases

1 INTRODUCTION

Relational databases are the most common way of organising data. However, they pose severe limitations for data handling because of their static set of tables with a fixed set of columns, each table representing a different entity or concept [8, 9, 11, 34, 35, 43]. There-

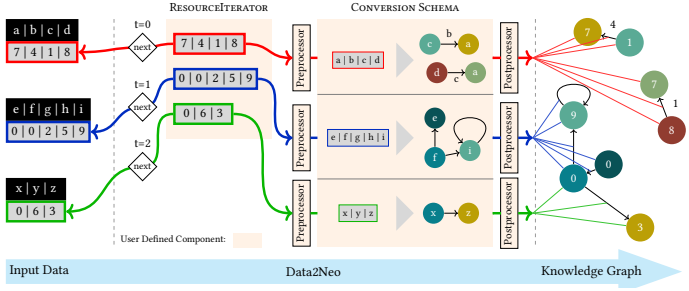
Generation (RAG) systems, by enhancing the integration and retrieval of structured knowledge. They provide context and background knowledge that is essential for sophisticated applications, such as question answering and advanced automated reasoning across various domains [10, 13, 15, 20, 23, 29, 31, 37, 39, 41]. In addition, they serve as a rich source of semantically structured information for high-quality training data [5, 27, 40]. Beyond their role in machine learning frameworks, KGs also have intrinsic potential for data analysis in their own right. They enable the extraction of insights through the relationships they map between different data points, which is critical in domains such as biomedicine [26, 42], cybersecurity [17, 22], and financial services [18]. The capacity of KGs to seamlessly integrate diverse data types and provide a holistic view makes them invaluable for predictive analytics and decision-making processes.

While certain datasets may opt for graph databases as their storage solution from the outset, there is considerable value in integrating existing relational data into knowledge graphs. However, integrating data into KGs might not be straightforward for many interested users, especially when the data integration necessitates employing complicated data transformation pipelines, e.g. when the data needs to be cleaned up or updated in real-time. To tackle these challenges, and to lower the entry barrier of data integration, we introduce **Data2Neo**[1], an open-source Python library designed for building data pipelines that convert relational data into knowledge graphs. The library provides extensive customization options while

Data2Neo: A tool for complex Neo4j data integration (2024).

Idea of data2neo

- Define a **conversion schema** (YAML style)
- Use built-in **iterators** (Pandas, SQLite) or custom ones for your data source.
- Supports **preprocessing** and **postprocessing** of attributes/relationships.
- Allows **batching**.
- Enables **incremental updates** / streaming import.



Minder, J., Brandenberger, L., Salamanca, L., and Schweitzer, F. (2024). Data2neo–a tool for complex neo4j data integration. *arXiv preprint arXiv:2406.04995*

Data2Neo by Julian Minder

Github page

`https://github.com/jkminder/data2neo`

- Check out the latest development
- Report issues

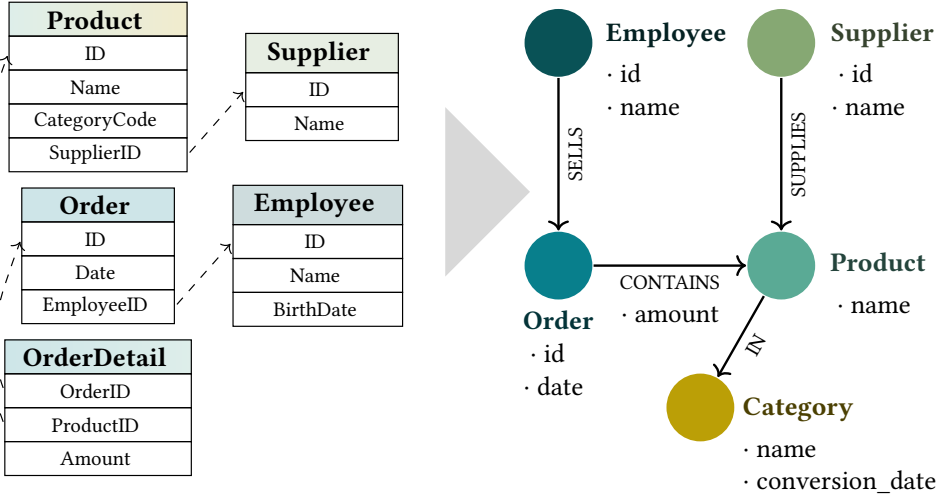
Documentation

`https://data2neo.jkminder.ch`

- This is the extensive documentation of the package
- Explains all elements of `data2neo`

Let's Run a Simple Example

- We'll import the Northwind data table



Import packages and setup

```
import neo4j
import pandas as pd

from data2neo.relational_modules.pandas import PandasDataFrameIterator
from data2neo import IteratorIterator
from data2neo import Converter
from data2neo.utils import load_file
from data2neo import register_subgraph_preprocessor

import data2neo.common_modules.types # For FLOAT, INT, etc. wrappers

import logging
```

- Import core and helper modules from `data2neo`.
- `PandasDataFrameIterator` helps read tabular data into the graph.
- `Converter` **performs the actual mapping to nodes and relationships**.
- The type wrappers (`FLOAT`, `INT`) ensure consistent property formats.

Sidenote: Logging

```
# Configure Logging
import logging

#logging.basicConfig(level=logging.WARNING)
logger = logging.getLogger("data2neo")
logger.setLevel(logging.INFO)
log_formatter = logging.Formatter("%(asctime)s [%(threadName)s]::[%(levelname)s]::%(filename)s: %(
    message)s")
console_handler = logging.StreamHandler()
console_handler.setFormatter(log_formatter)
logger.addHandler(console_handler)
```

- **Logging** creates structured, timestamped output instead of print statements.
- Makes it easy to trace what `data2neo` is doing — which entities are processed, warnings, etc.
- Not required, but extremely useful for debugging and long-running imports.

Defining a Schema in `data2neo`

```
schema = """
ENTITY("orders"):
  NODE("Order") order:
    + orderID = INT(orders.OrderID)
    - shipName = orders.ShipName
  RELATIONSHIP(order, "CONTAINS", MATCH("Product", productID = INT(orders.ProductID))):
    - unitPrice = FLOAT(orders.UnitPrice)
    - quantity = FLOAT(orders.Quantity)
"""
```

- A **schema** describes how tables and columns map to nodes and relationships.
- `ENTITY("orders")` defines one import block.
- `NODE` creates labeled graph nodes, and `RELATIONSHIP` connects them.
- The `+` marks a **primary key**, while `-` denotes a property.

Schema Example: Orders, Suppliers, and Products

```
ENTITY("suppliers"):
  NODE("Supplier") supplier:
    + supplierID = INT(suppliers.SupplierID)
    - companyName = suppliers.CompanyName

ENTITY("products"):
  NODE("Product") product:
    + productID = INT(products.ProductID)
    - productName = products.ProductName
    - unitPrice = FLOAT(products.UnitPrice)

  RELATIONSHIP(MATCH("Supplier", supplierID = INT(products.SupplierID)),
               "SUPPLIES", product):

  RELATIONSHIP(product, "PART_OF",
               MATCH("Category", categoryID = INT(products.CategoryID))):
```

- Each `ENTITY` can define multiple nodes and relationships.
- You can reuse nodes (e.g., `Product`, `Supplier`) across entities.
- `data2Neo` automatically merges identical nodes based on matching keys (+).

Schema Example: Employees and Categories

```
ENTITY("employees"):
  NODE("Employee") employee:
    + employeeID = INT(employees.EmployeeID)
    - firstName = employees.FirstName
    - lastName  = employees.LastName
    - title     = employees.Title
  IF_HAS_BOSS(
    RELATIONSHIP(employee, "REPORTS_TO",
                 MATCH("Employee",
                       employeeID = INT(employees.ReportsTo))))
ENTITY("categories"):
  NODE("Category") category:
    + categoryID   = INT(categories.CategoryID)
    - categoryName = categories.CategoryName
    - description  = categories.Description
```

- Conditional logic like `IF_HAS_BOSS` lets you define recursive relations.
- Great for hierarchical structures (e.g., managers, nested categories).
- Everything is written in a Python multi-line string, passed to the converter.

Create Custom Functions for Your Data

```python
@register_subgraph_preprocessor
def IF_HAS_BOSS(resource):
    if pd.isna(resource["ReportsTo"]):
        return None
    return resource
```

- **Custom preprocessors** let you add conditional logic before data is turned into nodes and relationships.
- The decorator `@register_subgraph_preprocessor` makes the function available inside your schema.
- Here, `IF_HAS_BOSS` checks whether an employee has a manager (`ReportsTo`) before creating a `REPORTS_TO` edge.
- Returning `None` means "skip this relationship" — prevents empty or invalid links.

Connect to Your Neo4j Database (Empty One)

```
uri = "neo4j://127.0.0.1:7687"
auth = neo4j.basic_auth("neo4j", "password")  # CHANGE TO YOUR CREDENTIALS

# OPTIONAL: start from a clean database
driver = neo4j.GraphDatabase().driver(uri, auth=auth)
with driver.session() as session:
    session.run("MATCH (n) DETACH DELETE n")
```

- Use this to connect your Python session to Neo4j.
- The delete command is **optional** → it clears all nodes and relationships.
- Only run it if you want to **start from an empty graph**.
- Skip it when adding data to an existing knowledge graph.

Create an Iterator

```
# Create IteratorIterator
files = ["categories", "employees", "orders", "products", "suppliers"]
iterators = []

for file in files:
    df = pd.read_csv(
        f"https://raw.githubusercontent.com/neo4j-documentation/"
        f"developer-resources/gh-pages/data/northwind/{file}.csv"
    )
    iterators.append(PandasDataFrameIterator(df, file))

iterator = IteratorIterator(iterators)
```

- Each CSV file is turned into a `PandasDataFrameIterator` — a wrapper that Data2Neo can read from.
- `IteratorIterator` combines multiple iterators into one unified data stream.
- This structure lets `data2neo` process multiple tables (entities) together during import.
- Works for any tabular source (CSV, SQL, Excel, Pandas).

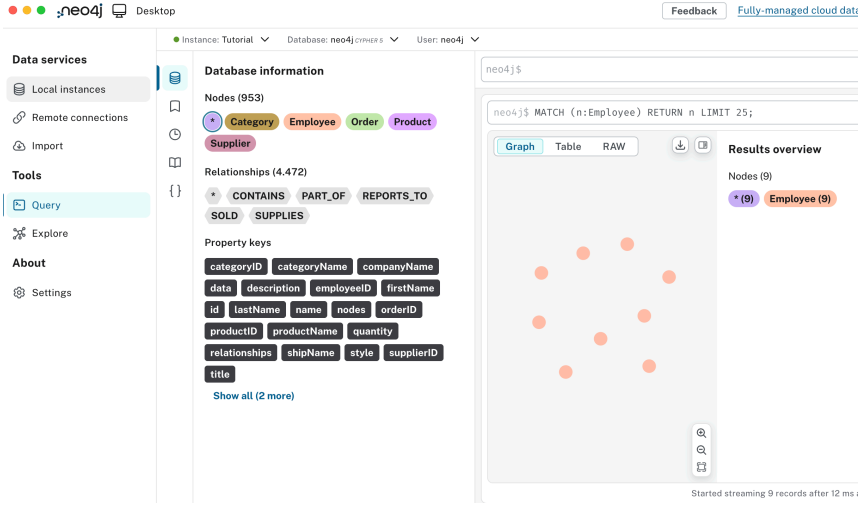
Hand It to the Converter

```
converter = Converter(
    schema,
    iterator,
    uri,
    auth,
    num_workers=1,
    serialize=True
)
from tqdm.notebook import tqdm
converter(progress_bar=tqdm)
```

- The `Converter` takes your **schema**, **data iterators**, and **Neo4j connection**.
- It parses the schema (YAML-like syntax) and automatically creates nodes, properties, and relationships.
- `num_workers` controls parallelism — increase it for faster imports.
- `serialize=True` ensures data are processed in a reproducible order (safe for demos).
- This is where your tabular data finally becomes a **knowledge graph**.

Check what you got in Neo4jDesktop

- Now head over to neo4jDesktop and check on your KG



Part E

Hands-On: The extensive example: Loading in UK Hansard data

or

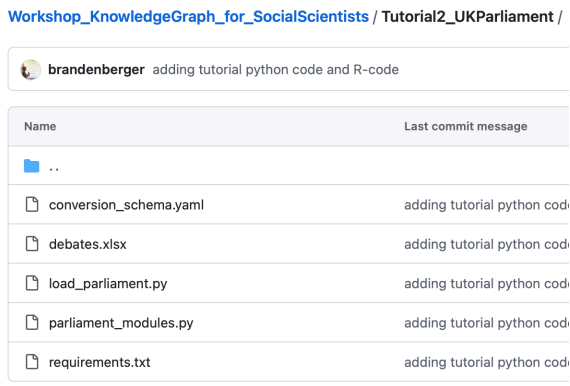
Prepared by Yaren Durgun

A More Realistic Example: Messy Data and Flawed Links

- Real-world data (like parliamentary records) are **incomplete and inconsistent**.
  - E.g., person names and parties don't always match up.
  - Multiple entries for the same person under slightly different spellings.
  - Departments, subjects, and member roles may be missing.
- Our wrappers and preprocessors help handle such cases:
  - Skip missing or malformed data.
  - Reuse existing nodes instead of creating duplicates.
  - Reconstruct missing links (e.g., member–party) when possible.

Code Organization: Keeping Things Modular

- `load_parliament.py` — main script
  - Loads Excel data.
  - Creates person-party affiliation map.
  - Sets up `PandasDataFrameIterator` and `Converter`.
- `parliament_modules.py` — wrappers and preprocessors
  - Handles data cleaning, caching, and relationship logic.
  - Registered via decorators (e.g. `@register_wrapper`).
- `conversion_schema.yaml` — schema definition
  - Maps Excel columns to graph nodes and relationships.



Modular design = easier debugging and reuse across datasets.

Custom Wrappers: Handling Complex Relationships

- Wrappers extend Data2Neo's `SubgraphFactoryWrapper` to handle custom graph logic.
- Example 1: **STORE_CHAMBER**
  - Ensures every debate links to the correct parliamentary chamber.
  - Reuses existing `Chamber` nodes instead of creating duplicates.
- Example 2: **BUILD_TEXTS_AND_LINKS**
  - Creates `Text` nodes for questions, answers, and debates.
  - Connects them to `Person`, `Party`, and `Debate` nodes.
  - Skips malformed or missing speaker data (e.g., "Speaker" entries).
- Wrappers let you manage caching, reusability, and graph consistency.

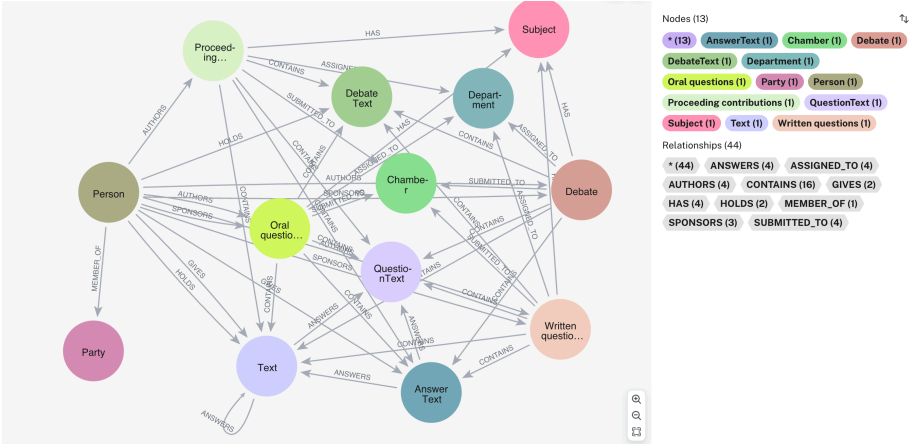
Global Shared State: Managing Cross-File Consistency

```
# In load_parliament.py
GlobalSharedState.affiliations = affiliation_map
```

- Stores a best-effort **affiliation map** of persons parties.
- Accessible to all wrappers during conversion.
- Enables fuzzy linking even when data are incomplete:
  - “John Smith” appears as “J. Smith” elsewhere still linked.
  - Missing party in one row can be inferred from others.
- Centralized shared state helps coordinate caching, matching, and deduplication.

The UK Parliament Mini-KG

- The schema of the UK Parliament KG



From Clean Imports to Real Data Challenges

- **Example 1 (Northwind):** showed the basics of building a knowledge graph
  - Clean tabular data clear one-to-one schema mapping
  - Learned how to define entities, relationships, and custom functions
  - Perfect to understand the `data2neo` workflow

- **Example 2 (Parliament):** reflected real-world research data
  - Messy and incomplete datasets require careful preprocessing
  - Modular design keeps data, logic, and schema separate
  - Wrappers and shared state handle exceptions and maintain consistency

- **Takeaway:** Knowledge graphs grow with your data
  - Start small — design cleanly, validate often
  - Add complexity only when needed
  - Embrace imperfection: structure and transparency are the real goals

Your data will never be perfect → but your graph can make sense of it.

Part E

Querying a Knowledge Graph: Basic Cypher queries

or

And a short intro to linking your data to your R environment.

Querying the Northwind Knowledge Graph: Basics

```
MATCH (p:Product)
RETURN p.productName, p.unitPrice
LIMIT 10;
```

```
MATCH (e:Employee)-[:SOLD]->(o:Order)
RETURN e.firstName, e.lastName, COUNT(o) AS orders
ORDER BY orders DESC
LIMIT 5;
```

- Retrieve nodes and their properties.
- Count how many orders each employee sold.
- Use `ORDER BY` and `LIMIT` for quick summaries.

Finding Connections in Northwind

```
MATCH (s:Supplier)-[:SUPPLIES]->(p:Product)
RETURN s.companyName, COUNT(p) AS numProducts
ORDER BY numProducts DESC;
```

```
MATCH (e:Employee)-[:SOLD]->(o:Order)-[:CONTAINS]->(p:Product)
RETURN e.lastName AS Employee, COUNT(DISTINCT p) AS ProductsSold
ORDER BY ProductsSold DESC;
```

- Combine multiple relationships with pattern chaining.
- Aggregate and summarize product-supplier or employee-product relationships.

Discovering Patterns and Filtering

```
MATCH (p:Product)-[:PART_OF]->(c:Category)
WHERE c.categoryName = "Beverages"
RETURN p.productName, p.unitPrice
ORDER BY p.unitPrice DESC;
```

```
MATCH (s:Supplier)-[:SUPPLIES]->(p:Product)-[:PART_OF]->(c:Category)
RETURN c.categoryName, COUNT(DISTINCT s) AS numSuppliers;
```

- Use `WHERE` for filtering and conditional selection.
- Combine categories, products, and suppliers in a single query.

Mini Challenge: Who Sells What?

```
MATCH (e:Employee)-[:SOLD]->(o:Order)-[:CONTAINS]->(p:Product)-[:PART_OF]->(c:Category)
RETURN e.lastName AS Employee, c.categoryName AS Category, COUNT(o) AS NumOrders
ORDER BY NumOrders DESC
LIMIT 10;
```

- Combine multiple paths to analyze who sells the most in each category.
- Great practice for reading multi-hop graph patterns.
- Try tweaking: filter by category, or group by both employee and supplier.

Exercise: Exploring the UK Parliament Knowledge Graph

Goal: Use Cypher to explore and analyze debates, parties, and text content.

1. **Easy – Connect & Inspect**
   - Connect to your `UK_Parliament` Neo4j database.
   - Use Cypher to list all available labels and relationship types.
2. **Medium – Speakers per Debate**
   - Retrieve all debates together with the number of MPs who authored or sponsored them.
3. **Medium+ – Active Departments**
   - Count how many debates each department has been assigned to.
   - Return the top 10 departments.
4. **Hard – Cross-Party Interactions**
   - Identify pairs of parties whose members participated in the same debate.
   - Count how many debates each pair co-appeared in.

Tip: Use `LIMIT 10` while testing queries to avoid huge results.

Exercise: Example Solutions

1. Inspect the schema

```
CALL db.labels();
CALL db.relationshipTypes();
```

2. Authors per debate

```
MATCH (d:Debate)<-[:AUTHORS|SPONSORS]-(p:Person)
RETURN d.title, COUNT(DISTINCT p) AS n
ORDER BY n DESC
LIMIT 10;
```

3. Active departments

```
MATCH (d:Debate)-[:ASSIGNED_TO]->(dept:
    Department)
RETURN dept.name, COUNT(DISTINCT d) AS n
ORDER BY n DESC
LIMIT 10;
```

4. Cross-party participation

```
MATCH (a:Party)<-[:MEMBER_OF]-(p1:Person)
      -[:AUTHORS|SPONSORS]->(d:Debate)
      <-[:AUTHORS|SPONSORS]-(p2:Person)
      -[:MEMBER_OF]->(b:Party)
WHERE a <> b
RETURN a.name, b.name, COUNT(DISTINCT d) AS n
ORDER BY n DESC
LIMIT 10;
```

Challenge: Group results by chamber or year.

Bonus Challenge: Topics by Party

Goal: Discover which political parties most frequently engage with specific topics.

- Find which topics (subjects) are most often linked to debates that members of each party participate in.
- Think of the traversal:

```
(Party) → (Person) → (Debate) → (Subject)
```

- Count how often each party–topic pair appears.
- Sort by the most frequent topics per party.

Hint: use `COUNT(DISTINCT ...)` *and consider combining relationships with* `[:AUTHORS|SPONSORS]`.

Bonus Challenge: Solution

```
MATCH (party:Party)<-[:MEMBER_OF]-(mp:Person)
      -[:AUTHORS|SPONSORS]->(d:Debate)-[:HAS]->(s:Subject)
RETURN
    party.name AS Party,
    s.name AS Topic,
    COUNT(DISTINCT d) AS Debates
ORDER BY Party, Debates DESC
LIMIT 20;
```

- Traverses across four node types to connect parties to subjects.
- Uses `COUNT(DISTINCT d)` to avoid double-counting the same debate.
- Produces interpretable party–topic pairs:
  - e.g. “Labour Healthcare (32 debates)”
  - e.g. “Conservative Taxation (27 debates)”
- Great exercise for multi-hop reasoning and interpreting graph data analytically.

Part F

Closing

or

I hope this workshop gave you a good foundation for knowledge graphs

Workshop Closing: Reflect & Discuss

Questions to Think About:

- Where in your research could a Knowledge Graph add value?
  - Integrating fragmented or historical data?
  - Tracking evolving relationships over time?
- What data structures or topics in your field could benefit from being *connected*?
- How might you use KGs to make your research more transparent and reusable?
- What challenges do you foresee when moving from spreadsheets to graph databases?

Thank you very much

Dr. Laurence Brandenberger

Senior scientist
University of Zurich
laurence.brandenberger@ipz.uzh.ch

Dr. Sophia Schlosser

Post-Doc
University of Zurich
sophia.schlosser@ipz.uzh.ch

Prof. Dr. Dr. Frank Schweitzer

Professor
ETH Zurich, Chair of Systems Design
fschweitzer@ethz.ch

Dr. Luis Salamanca

Lead Data Scientist
Swiss Data Science Center
luis.salamanca@sdsc.ethz.ch

Yaren Durgun

MA Student
University of Zurich
yaren.durgun@uzh.ch

Julian Minder

PhD Student
EPFL, Data Science Lab
julian.minder@epfl.ch

Research Assistants and Students: Yaren Durgun*, Nils Grob*, Rebecca Kessler*, Carina Pfister*, Yumi Kim, Kourosh Shariat, Dania Sana, Julian Minder, Leon Babic, Vincent Jung, Marta Balode, Teodora Bujaroska, Sarolta Gabulya, Jordi Campos, Daria Izzo, Clemens Hutter, Anna Schmitt-Rohr
[*Currently active]

References I

Brandenberger, L., Schlosser, S., Salamanca, L., Gasser, L., Balode, M., Minder, J., Jung, V., Durgun, Y., Babic, L., Perez-Cruz, F., and Schweitzer, F. (2025). Democrasci: A knowledge graph on the swiss parliament. *Paper submitted to Nature. Data Descriptor*.

Minder, J., Brandenberger, L., Salamanca, L., and Schweitzer, F. (2024). Data2neo–a tool for complex neo4j data integration. *arXiv preprint arXiv:2406.04995*.

Salamanca, L., Brandenberger, L., Gasser, L., Schlosser, S., Balode, M., Jung, V., Perez-Cruz, F., and Schweitzer, F. (2024). Processing large-scale archival records: The case of the swiss parliamentary records. *Swiss Political Science Review*, 30(2):140–153.

Salamanca, L., Meyer, M., Brandenberger, L., Schlosser, S., Campos-Schweitzer, J., Schweitzer, F., and Perez-Cruz, F. (2023). pdf2data: A tool for extracting information from pdf documents.