Graph lenses over any data: the ConnectionLens experience

Oana Balalau, Nelly Barret, Simon Ebel, Théo Galizzi, Ioana Manolescu, Madhulika Mohanty

Graph lenses over any data: the ConnectionLens experience

Oana Balalau, Nelly Barret, Simon Ebel, Théo Galizzi, Ioana Manolescu, Madhulika Mohanty
Inria & Institut Polytechnique de Paris
Email: firstname.lastname@inria.fr

***Abstract*—Data integration is decades-old problem that takes many shapes, depending on the model of the integrated data sources, the integration model (if a single model is used), the expressivity of the features supported from each model, etc.**

Over several years, we have worked to integrate very heterogeneous data, aiming to address the needs Non-Technical Users (NTUs), notably journalists. The choice we make is to integrate data of any model by migrating (transforming) it into a *graph*, consisting simply of labeled nodes and edges. Such graphs are much simpler than Property Graphs and more basic even than RDF graphs, since we do not require URI labels on internal nodes. This experience paper gives an overview of our research efforts towards querying, understanding, and exploring the resulting graphs. The contributions we brought are in the areas of data integration, graph exploration, graph querying, and go towards managing semistructured data lakes.

I. Motivation: data integration for NTUs

Increasing volumes of digital data sources are being shared at large scale, sometimes following W3C's best practices for Open Data (that is, as RDF graphs), but also often in other formats, such as CSV, Office formats, XML, HTML, JSON, etc. This creates opportunities to develop applications that integrate data from heterogeneous data sources, fulfilling information needs that cannot be answered from one data source alone. In a traditional data integration setting, four main questions must be answered: (i) How to *model* the target (integrated) data? (ii) How to *express* the connections between the data sources and the integrated model? (iii) How to *formulate queries* over the integrated view, and (iv) How to *efficiently process* such queries?

Most research on data integration assumed a relational data model for the sources, and also for the integrated schema (question (i)). This setting has allowed researchers to study fundamental questions such as the relationships between the local and global relational schemas [17], and how to migrate data from one model to another via schema mappings [10]. Subsequently, this setting has been extended, with sound logical foundations, to Ontology-Based Data Access (OBDA, in short) [14], [19], where the global schema and integrity constraints over it are described in a dialect of Description Logics; this area of work has fruitfully interacted with Semantic Web standards to integrate data as RDF graphs to which we associate an ontology, describing the semantics of the integrated application, e.g., [11], [12], [13].

However, when data integration is needed by **non-technical users (NTUs, in short)**, especially if the data sources are not relational (which is often the case outside of an enterprise setting), setting up such an integration scenario requires skills and resources that NTUs do not have. Also, adapting the solution would require too much work when the set of data sources changes.

This is why, in late 2017, inspired by the Panama Papers project journalistic investigation (subsequently renamed into Paradise Papers)[1], we took a different path. Specifically, we decided to **integrate any set of heterogeneous datasets into a directed graph**, consisting of nodes and edges, each carrying a label (that could also be empty). Thus, our graphs are more simple than RDF [21] standard graphs, notably because we do not require or assume each node to have a distinct URI, nor a label. This allows us to directly integrate data from models where labels are lacking from nodes, e.g., JSON internal nodes, or from edges, e.g., parent-child relationships between XML elements. The approach we take is to assign simple integer IDs to all the nodes ingested in a graph, while preserving for each its label (empty or not), and the dataset from which it originates. Transforming our data graphs to RDF is syntactically straightforward, and we do not discuss it here. Further, we enrich the ingested data graph with the help of Named Entity Recognition: whenever a named entity is identified within a text field, we model it as a new node, child of each text field in which it has been recognized. Thus, named entity nodes allow interconnecting datasets, leading to a form of graph-mediated integration.

Below, we outline: how we transform of various kinds of data in directed graphs, leveraging also the help of Named Entity Recognition (Sec. II); how to search and query such highly heterogeneous graphs (Sec. III); how we simplify these graphs for NTUs, into dataset abstractions, akin to Entity-Relationship diagrams (Sec. IV); how NTUs can inspect their content, through user-friendly query interfaces and entity paths (Sec. V). We end with a discussion and some open issues.

II. Integrating heterogeneous data into graphs

The ConnectionLens system [3], [15] ingests structured, semi-structured, and unstructured data into a **simple directed graph**. Each node and edge has a label from a set of labels $\mathcal{L}$ (including the empty label ϵ); each node is assigned a numeric ID within the graph. The ingestion *preserves all the structure from the original dataset*, e.g., relationships

[1] https://www.icij.org/investigations/paradise-papers/

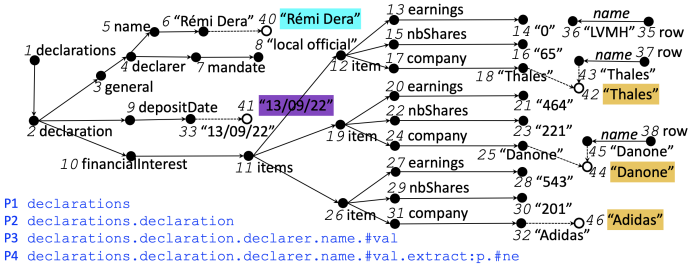

Fig. 1. Sample data graph built from HATVP declarations and CAC40 companies (from [5]).

between nodes and their children, tuples and their attributes, etc. Each XML element, attribute, or text node becomes a graph node; parent-child relationships in the XML document lead to corresponding edges in the graph. A JSON document is similarly converted: each map, array, and (leaf) value is converted into a graph node. RDF graphs are most easily ingested: each triple of the form ⟨s p o⟩ leads to two nodes labelled "s" and "o" connected through an p-labelled edge. For CSV and relational data, each tuple and value lead to a node, edges labelled with the column names are connecting those (if the column name is empty, so the edge label). Text documents are segmented into paragraphs, each of which is a node, child of a common root. Office and PDF documents are converted into JSON, then ingested as above.

NER (Named Entity Recognition) is applied on every leaf node of the graph, leading to (new) extracted entity nodes. Each entity node is labelled with the recognised **named entity** (**NE**, in short) and modeled as a child of the text node from which it has been extracted. When two NE nodes are identical, i.e., they have same type and label, they are fused: the second time a NE is encountered in a string, the node created from its first appearance is connected also as a child of the second string. This leads to connections *across datasets*. The currently supported entity types are: Person, Location, Organization, date, URI, email, hashtag and mention. To extract entities, we use the StanfordNLP [20], Flair [1], and, most recently, the ChatGPT v4 API. Each is faster, and of lower quality (more false positives and negatives), than the next one listed. The first two are free, while the last incurs financial costs.

Fig. 1 shows the graph resulting from two datasets thus ingested. The first one (at left in Fig. 1) is an XML dataset produced by the HATVP (a French state authority) describing ministers' wealth declarations. The second (bottom right) is a CSV file listing the 40 most influential companies in France (known as "CAC 40"). Internal nodes are shown as white boxes, value nodes as grey boxes. Coloured boxes are extracted NEs: green for places, purple for dates, yellow for organizations, and blue for people. The NE nodes 43 "Thalès" and 42 "Danone" have two incoming edges: this is because the respective NEs have been identified in two different strings, one in the HATVP, the other in the CAC40 datasets.

The cost of ingesting data into a ConnectionLens graph has two components: (i) building and storing (in Postgres) the graph nodes and edges; this is linear in the number of nodes, respectively, edges; (ii) extracting named entities from the text nodes present in the graph; this is linear in the total size of the text nodes of the graph. The latter task is much more expensive than the former (from a factor of 2 to one order of magnitude), thus, NE extraction dominates the graph creation costs. Batch extraction (sending several strings together for entity extraction) allows to take advantage of the multiple cores available in CPUs and/or GPUs in order to significantly decrease the loading time. NE extraction, however, still remains the bottleneck [3].

III. Searching and querying simple graphs

The structure of simple data graphs thus obtained may be more or less regular, depending on the number and the regularity of the input datasets. Such graphs, where each node has just a label, can be seen as a simplified version of RDF graph (with no blank nodes, and no ontologies). Thus, a SPARQL-like language, based on query triples, can be used to query them. However, expressing queries is hard for NTUs; and even for experts, lack of knowledge about the data structure makes a structured query language hard to leverage. This is why we have, first, developed **keyword search algorithms** which, when given some keywords, such as "Danone" and "Dera", on the graph in Fig. 1, return all the paths connecting two nodes, each having a label matching one of the keywords. When there are more than two keywords, the answer is more generally a (minimal) tree, having a node matching each keyword. We have proposed GAM (Grow and Aggresive Merge), an exhaustive algorithm to answer such queries [3], then several optimized versions, much faster by pruning intermediary trees, and formalized their guarantees [4]. Importantly, *any scoring function can be used* to rank answers, as our algorithms are totally *independent (orthogonal) from the score*. However, their complexity remains high; on dense and/or large data graphs, a time-out and/or a bound on answer size should be set. Journalists (our primary intended NTUs) found such limits reasonable: in large graphs, long paths may end up connecting any pair of nodes, making the path quite meaningless.

This work lead us to seek to advance the integration of keyword search into structured graph querying, à la SPARQL and the PG graph pattern matching language GPML [16]. A query language like SPARQL allows to ask for arbitrary paths between two nodes, yet users must specify a regular expression over edge labels which the path should satisfy. In GPML, completely unspecified *paths* can be returned, but not trees. Going beyond, in [4], we also proposed an Extended Query Language, **blending conjunctive graph querying à la GPML, with keyword search based on any number of keywords**. For instance, such a query allows finding any connection between a node having a name child whose value is "Rémi Dera", a date in 2022, and "Thalès". This language, demonstrated recently in a SPARQL-like variant [2], is technically interesting, but too complex for NTUs to use.

IV. Abstracting (semi-)structured datasets

The fine-granularity simple graphs are suited for search via keywords. But how does one "zoom out" to *get a first glimpse*

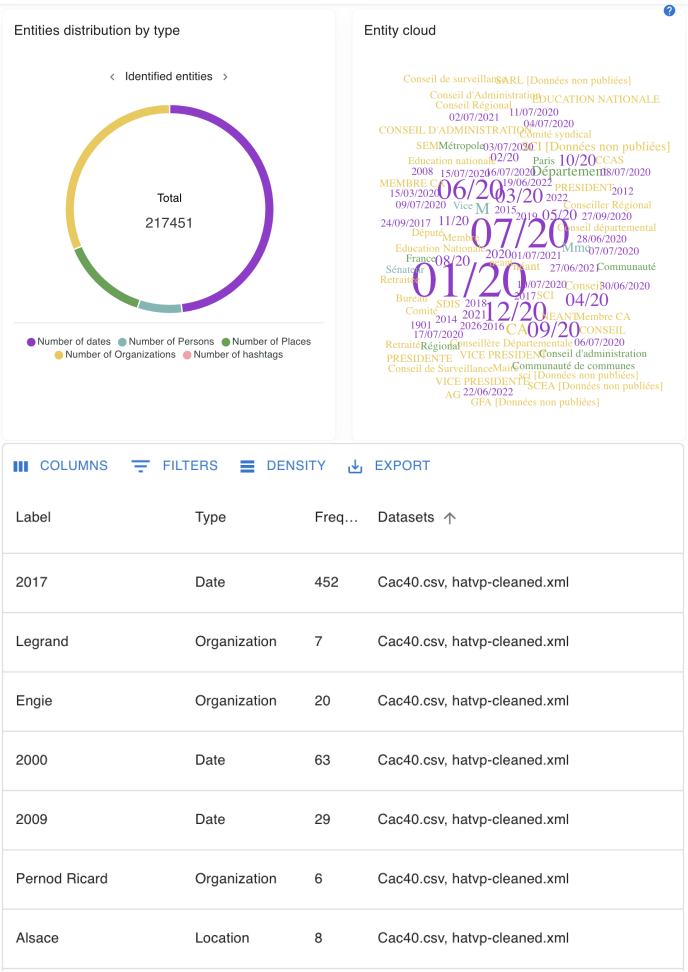

Fig. 2. Entity/dataset statistics for the sample dataset (from [5]).

of an entire dataset?

In the ABSTRA project [8], [9], to answer this question, we **automatically derive from the data graph** an **abstraction (Entity-Relationship like diagram)**, providing users a first intuitive, visual representation of the data. We did this as follows.

1) We build a *structural quotient summary* of the data graph, based on partitioning the nodes into *equivalence classes*, and creating one summary node for each group of equivalent nodes. We consider equivalent structural (non-leaf) nodes that belong to the same data model, e.g., an RDF URI is only equivalent to an RDF URI, an XML element to an XML element, etc. To decide which nodes are equivalent, we rely on different notions for each data model, e.g., XML elements with the same label, or more elaborated notions of equivalence for RDF [18]. The summary of a simple graph is a directed graph itself; each of its nodes is called a *collection* (of simple graph nodes), and there is an edge from a collection to another, if a corresponding edge existed in the simple graph. We call the summary a *collection graph*.
2) Among the *node collections* identified by summarization, a collection may contain nodes which are "roots" of structured entities, e.g., person or company nodes; we call such a summary node, a *main entity node*. Other collections may describe internal properties, or attributes, of main entities; such properties may have an internal nested structure. We call *boundary* of a main entity, a set of nodes reachable from the main entity nodes, and which, together, describe all the internal structure of the main entity. We proposed several algorithms for automatically identifying main entities and their boundaries [9] and show that the best-performing succeed in capturing the natural semantic of the dataset.
3) Having identified the main entities, paths (in the collection graph) that lead from one main entity to another are interpreted as *relationships* among them.
4) Beyond identifying each entity's structure, we also attempt to *classify* them using a set of classes with easily understandable names, e.g., `Person`, `Project`, `CreativeWork`, etc. For this task, we leverage linguistic and semantic resources, as well as the names of the properties that each entity has [9].

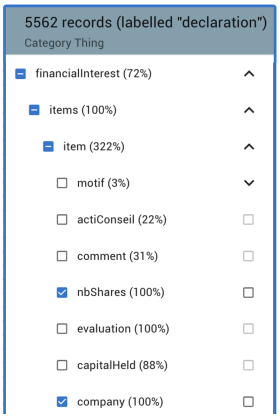

Fig. 3. Querying the data lake through data abstractions (from [5]).

The ABSTRA code, and a gallery of sample abstractions, are available online at https://team.inria.fr/cedar/projects/abstra/.

V. A TOOLBOX FOR EXPLORING INTEGRATED GRAPHS

Once users (and in particular, NTUs) have grasped the high-level structure of the data via abstractions, how to entice, then assist them in exploring it?

We built CONNECTIONSTUDIO [5], a new interactive front-end exposing: the simple graph and its keyword search; the abstractions; and a set of novel features we describe below.

A. Entity/dataset statistics

We compute and show a set of *entity* and *entity-dataset statistics*, as follows: the total numbers of NEs of each type in the integrated graph; the distribution of NEs per type and dataset; a tag cloud of the most frequent NEs; and *a summary of the entity-dataset associations*. The latter shows the entity label, type, and datasets where it appears, starting with the entities present in the highest number of datasets. These enable potentially interesting connections *across* data

sets (and possibly across data models), insights that cannot be gained without an integrated approach. These statistics also suggest entity names to use as search keywords (Sec. II).

B. Querying with elementary paths

Upon loading, CONNECTIONSTUDIO computes, from each dataset, a set of *elementary paths* reflecting the dataset structures. Each elementary path p is a sequence of alternating node and edge labels. The source node of a path (n_1) always corresponds to an internal data node, while its destination (n_k) is either an internal node, a value, or a named entity extracted from a value. For instance, some elementary paths are shown in blue at the bottom left of Fig. 1. `declarations` and `declarations.declaration` end up in internal nodes; `declarations.declaration.declarer.name.#val` ends in politician names; `declarations.declaration.declarer.name.#val.extract:p.#ne` ends in Person NEs extracted from these strings (recall Sec. II); note the special extraction edge label `extract:p`. From XML or JSON documents, we extract each path starting from the document root, and ending in an internal, value, or NE. From relational data, we extract each path starting in a tuple node and ending in a value or NE. From RDF, for each property `p` encountered in an $\langle \mathsf{s\ p\ o} \rangle$ triple, we extract simply `p` (formally $\epsilon.\mathtt{p}.\epsilon$) as an elementary path.

In the CONNECTIONSTUDIO GUI, choosing one dataset leads to a drop-down menu of its elementary paths, where the user may pick one or several. The first is "required"; the others may be required or "optional". Each path is attached a "start" and an "end" variables, which can be renamed by the user; reusing a variable name is an intuitive way to express a join. CONNECTIONSTUDIO converts a set of required or optional paths, whose end nodes are assigned (possibly shared) variable names, into a query q of the form $p_1 \circ_1 p_2 \ldots \circ_n p_n$ where each p_i is a path, each $\circ_i$ is either $\bowtie$ or $⟕$. Required paths are joined with p_1; optional paths are outer-joined with the join results. Because elementary paths end in either nodes, or values, or NEs, such queries may express *arbitrary structural patterns* (joining on nodes), as well as *joins on values or NEs across datasets* (and possibly different data models). For instance, we may want to extract: for each elected politician, their name, and CAC40 companies in which they may have investments (if any). This is achieved (Fig. 4) by joining four XML elementary paths, returning, respectively: the French officials' names; their declared financial interests (`item` elements); the companies in which they have stocks; the number of their shares in the company. To restrict the results to CAC40 companies, we add a fifth path (the last in Fig. 4) coming from the CSV dataset. This path is joined with the previous ones, on the common variable `companyName`.

C. Point-and-click querying based on abstractions

When shown a dataset abstraction, a user can select some entities, some of their attributes, and relationships connecting them, in point-and-click mode as shown by ticked attributes in Fig. 3. A graph pattern query is automatically built, extracting from the data graph the selected entity and/or relationship attributes.

D. Automatic identification of NE-to-NE paths

NTUs, and particularly journalists we discussed with, are interested in digital datasets if it can teach them something about their topics of interest, in particular: *what are the connections between NEs, that this dataset (or set of datasets) reveal?* In [6], we have proposed algorithms for **automatically identifying, in a CONNECTIONLENS graph, the paths connecting pairs of NEs** of user-chosen types τ_1, τ_2 (People, Location, Organization, etc.). For efficiency, we *identify* the paths based on ABSTRA's dataset summary (Sec. IV); this is usually orders of magnitudes smaller than the data, and easily fits in memory. We then *rank* these paths according to metrics we introduced, which (i) guard against spurious paths due to false-positive NEs, (ii) reward paths representing strong connections among nodes, e.g., an edge from a person to a spouse (out of very few in a lifetime) is stronger than an edge from a person to a friend (of which they have many).

CONNECTIONSTUDIO is available online at https://connectionstudio.inria.fr.

VI. DISCUSSION

The tools and methods we described have succeeded in getting journalist NTUs interested; we demonstrated them in several journalistic events, e.g., CFI'23. We realized the huge importance of an *integrated, simple GUI*, which we were able to develop only recently [5]. Used to Excel, journalists consistently favored *tabular* views of the data; our queries and automatically recommended NE paths (Sec. III, Sec. V) compute such tables from the simple graph.

Granularity We made the choice of simple nodes and edges, with just a (possibly empty) label. This facilitates ingestion since any data can be input; these data graphs are later interpreted, via abstraction, as consisting of entities with attributes (Sec. IV). In a companion paper [7], we leverage this understanding to transform any (semi-)structured data in Property Graphs.

Extraction, quality, matching NE extraction needs text inputs, e.g., "Joe Biden" as a string, not a structured (first, last) pair. We heuristically modify data upon ingestion, to produce such People name strings. Yet, the data cleaning literature shows that *internal record structure* helps when matching similar records. We currently compare NEs through label similarity, and leverage disambiguation to map an NE to a KB URI [3]; similar NEs, respectively, those considered identical, are connected with special *same-as* edges. We now consider adding *holistic* methods to match *structured* named entities. NE extraction quality matters a lot (Sec. II) for the graph quality, also to reduce erroneous NE matching effort.

Acknowledgments This work is partially funded by DIM RFSI PHD 2020-01, AI Chair SourcesSay (ANR-20-CHIA-0015-01) and CQFD (ANR-18-CE23-0003) grants. We also thank Camille Pettineo who contributed insights as a data journalist.

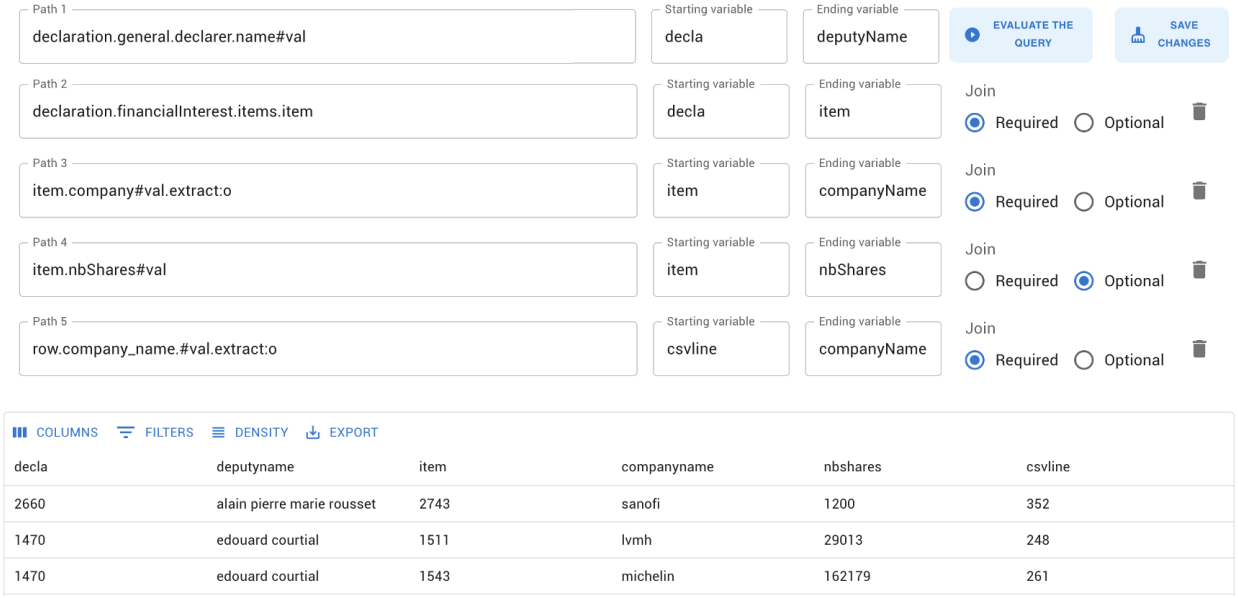

Fig. 4. Querying the data lake using elementary paths to know more about politicians' investments (from [5]).

REFERENCES

[1] Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. FLAIR: an easy-to-use framework for state-of-the-art NLP. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Demonstrations*, pages 54–59. Association for Computational Linguistics, 2019.

[2] Angelos Christos Anadiotis, Ioana Manolescu, and Madhulika Mohanty. More power to SPARQL: From paths to trees. In *ESWC*, 2023.

[3] Angelos-Christos G. Anadiotis, Oana Balalau, Catarina Conceição, Helena Galhardas, Mhd Yamen Haddad, Ioana Manolescu, Tayeb Merabti, and Jingmao You. Graph integration of structured, semistructured and unstructured data for data journalism. *Inf. Syst.*, 104:101846, 2022.

[4] Angelos-Christos G. Anadiotis, Ioana Manolescu, and Madhulika Mohanty. Integrating connection search in graph queries. In *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*, pages 2607–2620. IEEE, 2023.

[5] Nelly Barret, Simon Ebel, Théo Galizzi, Ioana Manolescu, and Madhulika Mohanty. User-friendly exploration of highly heterogeneous data lakes. In *Cooperative Information Systems - 29th International Conference, CoopIS 2023, Groningen, The Netherlands, October 30 - November 3, 2023, Proceedings*, volume 14353 of *Lecture Notes in Computer Science*, pages 488–496. Springer, 2023.

[6] Nelly Barret, Antoine Gauquier, Jia Jean Law, and Ioana Manolescu. Exploring heterogeneous data graphs through their entity paths. In *Advances in Databases and Information Systems - 27th European Conference, ADBIS 2023, Barcelona, Spain, September 4-7, 2023, Proceedings*, volume 13985 of *Lecture Notes in Computer Science*, pages 163–179. Springer, 2023. Extended version invited to Inf. Systems, under review.

[7] Nelly Barret, Ioana Manolescu, Madhulika Mohanty, and Tudor Enache. Finding the PG schema of any (semi)structured dataset: a tale of graphs and abstraction. In *SEAGRAPH workshop*, 2024.

[8] Nelly Barret, Ioana Manolescu, and Prajna Upadhyay. Abstra: Toward generic abstractions for data of any model. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*, pages 4803–4807. ACM, 2022.

[9] Nelly Barret, Ioana Manolescu, and Prajna Upadhyay. Computing generic abstractions from application datasets. In *Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28*, pages 94–107. OpenProceedings.org, 2024.

[10] Zohra Bellahsene, Angela Bonifati, and Erhard Rahm, editors. *Schema Matching and Mapping*. Data-Centric Systems and Applications. Springer, 2011.

[11] Elena Botoeva, Diego Calvanese, Benjamin Cogrel, Martin Rezk, and Guohui Xiao. OBDA over non-relational databases. In *Alberto Mendelzon Workshop*, 2016.

[12] Maxime Buron, François Goasdoué, Ioana Manolescu, and Marie-Laure Mugnier. Obi-wan: Ontology-based RDF integration of heterogeneous data. *PVLDB*, 13(12):2933–2936, 2020.

[13] Maxime Buron, François Goasdoué, Ioana Manolescu, and Marie-Laure Mugnier. Ontology-based RDF integration of heterogeneous data. In *EDBT*, pages 299–310, 2020.

[14] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. The MASTRO system for ontology-based data access. *Semantic Web*, 2(1), 2011.

[15] Camille Chanial, Rédouane Dziri, Helena Galhardas, et al. ConnectionLens: Finding connections across heterogeneous data sources (demonstration). *PVLDB*, 11(12), 2018.

[16] Alin Deutsch, Nadime Francis, Alastair Green, Keith Hare, Bei Li, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Wim Martens, Jan Michels, Filip Murlak, Stefan Plantikow, Petra Selmer, Oskar van Rest, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Fred Zemke. Graph pattern matching in GQL and SQL/PGQ. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 2246–2258. ACM, 2022.

[17] AnHai Doan, Alon Y. Halevy, and Zachary G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.

[18] François Goasdoué, Pawel Guzewicz, and Ioana Manolescu. RDF graph summarization for first-sight structure discovery. *The VLDB Journal*, 29(5):1191–1218, April 2020.

[19] Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.

[20] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*, pages 55–60. The Association for Computer Linguistics, 2014.

[21] W3C. RDF 1.1 Concepts and Abstract Syntax, 2014.