Finding the PG schema of any (semi)structured dataset: a tale of graphs and abstraction

Nelly Barret, Tudor Enache, Ioana Manolescu, Madhulika Mohanty

Finding the PG schema of any (semi)structured dataset: a tale of graphs and abstraction

Nelly Barret*, Tudor Enache†, Ioana Manolescu* and Madhulika Mohanty*
* Inria & Institut Polytechnique de Paris, France
{firstname.lastname}@inria.fr
† Ecole Polytechnique, France
tudor.enache@polytechnique.edu

***Abstract*—Property Graphs (PGs) are an attractive data model both for business users, and for developers of data management tools. They combine the internal structure helpful in relational databases, where each record has a clearly identified set of attributes, with the flexible structure and support for heterogeneity, common in graph databases.**

Several useful and/or interesting datasets are available in non-PG data models. These include legacy databases, created before the advent of the PG standards, as well as well-known benchmarks based on real and synthetic data, Open Data published in other formats such as XML, JSON or RDF, etc. Converting such datasets to Property Graphs would enable their exploitation under the PG model.

In this work-in-progress paper, we describe an approach to *derive, from any (semi)-structured dataset, a PG schema* consisting of node types, edge types, and a graph type. Our approach builds on (i) ConnectionLens, a tool for converting (semi)-structured datasets into simple data graphs, and (ii) Abstra, which, in a ConnectionLens graph, identifies a set of *entities* and *relationships*. This work is the first step towards a universal data migration tool from (semi)-structured data, to PGs.

I. Motivation and outline

There is an unprecedented creation of data pertaining to various contexts, like health, finance or social networks. Even though the W3C recommends sharing the data as RDF graphs, practitioners still use other models, e.g., relational, XML, JSON, etc. While semi-structured data models are very flexible and enable describing data with varied structure, datasets shared under these forms may be hard to understand for new users, especially if the data is shared with insufficient or no documentation. To solve this problem, there have been several efforts to *generate (infer)* a schema from the data itself. Existing schema generation approaches are each designed for a given data model, e.g., for JSON [4], [3], [16], [27], XML [10], and RDF [13]. In a similar but different vein, Abstra [5] generates, from a (semi-)structured dataset of one among multiple models, a *dataset abstraction*, akin to the traditional Entity-Relationship diagrams [23], but also allowing *deeply nested* entities. An abstraction is data model-independent; it is not a grammar, but a diagram, giving users a first look at the dataset.

More recently, **Property Graphs** (PGs, in short) are adopted in various application domains, e.g., the International Consortium of Investigative Journalism (ICIJ) built the Offshore leaks PG database [30], which has been used to detect tax evasion. Numerous industrial PG databases exist, e.g., from Neo4J [19] and Oracle [20]. Property graphs consist of labeled nodes, possibly connected by labeled edges; both nodes and edges may have attributes in the form of *key/value pairs*. This follows the "record-style" of existing relational databases (an object has a label and some attributes), while accounting for semi-structured data heterogeneity and complexity (not all records have the exact same set of attributes). Also, edges may have their own attributes, something that is not possible in relational model. The popularity of PGs has led to efforts to standardize the data model, the query language, and to generate a schema from a given graph [7], [16]. A recent, comprehensive proposal for PG schemas (in the classical database sense: schema defined independently of a dataset, introducing types that the dataset may or may not validate) is [2]. In order to be able to exploit these benefits, many prior works have targeted the problem of automatically converting a relational dataset into a graph [25], [28].

In this work-in-progress paper, given *any* semi-structured dataset, we aim to **derive a PG schema** for it. We achieve this in the following steps:

(i) Create a simple data graph out of the semi-structured dataset using ConnectionLens [1] (Sec. II-A);

(ii) Abstract this data graph to detect the entities and relationships it describes, using Abstra [5] (Sec. II-B);

(iii) Derive a PG type, as described in [2], from these entities and relationships (Sec. III). This is the main contribution of this paper.

We also evaluate the quality and soundness of the generated PG schemas on several (semi)-structured datasets (Sec. IV). Finally, we conlude and provide future extensions that could be built on top of our PG schema generator for (semi-)structured data (Sec. V).

II. Background

A. From (semi-)structured data to a data graph

ConnectionLens [1] (CL, in short) is a system which, starting from any (set of) structured, semi-structured or unstructured datasets, converts it to a *simple data graph*, with atomic nodes, and labeled nodes and edges. In this graph, nodes have a unique ID and a label (possibly empty), edges have a unique ID, a source node, a target node, and a label (also possibly empty). Thus, ConnectionLens constructs

$G = (N, E, \lambda)$ where N is the node set, $E \rightarrow N \times N$ is the edge set, and λ is a function labeling nodes and edges, possibly with the empty label ϵ. CL constructs a simple data graph as follows. XML documents translate into trees, where each element node, respectively element or attribute value leads to a node in G. Edges are modeling the parent-child relationships. An edge connecting an element node to an attribute value is labelled with that attribute name; other edges are labeled ϵ. When an XSD [31] accompanies the data, ID-IDREF connections lead to an edge between the IDREF node to the ID node, thus the resulting graph G is no longer a tree. JSON documents also lead to trees, where each map, array and (map or array) value is modelled as a node. A map node is connected to each of its attribute values by an edge labelled with the attribute name, while an array node is connected to its value using an ϵ-labelled edge. RDF graphs are easily converted to simple graphs: each triple $\langle s \rangle \langle p \rangle \langle o \rangle$ leads to a p-labelled edge connecting a node labelled s to a node labelled o. For CSV tables, a node is created for each line (tuple), respectively value. If a header was present, edges connecting lines to their value are labelled with the corresponding header name, otherwise the edge is ϵ-labelled. We call *value nodes* the data nodes created out of XML (element or attribute) values, JSON (attribute or array) value, RDF literals and CSV values. Those values are constants. Others nodes, e.g., XML elements, JSON map or array, etc. are *structural nodes* as they organize the data.

B. From a data graph to an abstraction [5]

To better understand a dataset, and towards identifying entities and their possible relationships, ABSTRA [5] builds an abstraction thereof, as follows. First, it *summarizes* the simple data graph G based on an equivalence relation among the nodes in the graph. The resulting summary is a (much smaller) graph $\mathcal{G}$, each of whose nodes corresponds to a set (or collection) of nodes from G; for each edge in G, $\mathcal{G}$ has an edge between the respective two $\mathcal{G}$ nodes. We call $\mathcal{G}$ a *collection graph*, and its elements *collection nodes*, respectively, *collection edges*. Each data model is summarized with the equivalence relation best suited to it. Thus, it considers equivalent: XML nodes having the same label, JSON and CSV nodes on the same path from the root. For RDF nodes, summarization relies on a flexible, type-and-structure-based equivalence relation introduced in prior work [12].

Next, ABSTRA selects a set of collection nodes $\mathcal{E} \subseteq \mathcal{G}$ to be promoted as *(main) "entities"*; the remaining $\mathcal{G}$ nodes will either be considered attributes of one or several entities in $\mathcal{E}$, or found to describe relationships between these entities. Users can limit the size of $\mathcal{E}$, in which case ABSTRA will reflect only the entities containing "most" data nodes. Without this limit, all the dataset is reflected in the returned entities, whose number depends on the dataset's structural complexity. For instance, out of an XMark [24] XML document of 5M nodes describing an auction website, given a limit of 5 entities, ABSTRA identifies: `open_auction`, `closed_auction`, `item`, `person` and `category` records (the five boxes in Fig. 1). A *boundary* is then computed for each such main entity: a set of $\mathcal{G}$ nodes considered to be part of (attributes belonging to) the main entity, and the edges connecting these nodes to each other, and to the main entity. While in classical E-R design [23] all entity attributes have atomic values, attributes of these entities can be nested. For instance, the boundary of the `person` entity includes: `name`, `emailaddress`, `id`, `homepage`, `phone`, `creditcard`, and `address`; the latter has the nested attributes `province`, `city`, `zipcode`, `country` and `street` (not shown in Fig. 1).

Third, to each main entity is assigned a *semantic class* from an ontology built based on open Knowledge Bases (KB) and other linguistic resources, leveraging the labels of the nodes in the entity (node collection) and/or the labels of their attributes. For instance, the `item` entity is classified as a `Product`, mainly because it is labelled `item`, it has `quantity` and `shipping` attributes. In the last step, a set of *relationships* $\mathcal{R}$ connecting the main entities is identified based on the $\mathcal{G}$ paths connecting the main entity nodes. For instance, an `item` has a `category` and an `open_auction` refers to an `item`.

C. PG Schema language [2]

We recall the recent PG schema [2] proposal on which we build our work. Let $\mathcal{L}$ be a set of (node and edge) labels; a PG schema graph type T_G consists of a set of node types T_N and a set of edge types T_E. A *node type* T_N^i specifies a certain set of node labels $L^i \subseteq \mathcal{L}$ and set of attributes $\mathcal{A}^i \subseteq \mathcal{A}$, the complete set of (node and edge) attributes; this is denoted $(T_N^i: L^i\ \mathcal{A}^i)$. An *edge type* T_E^j is characterized by a set of edge labels $L^j \subseteq \mathcal{L}$, a set of edge attributes $\mathcal{A}^j \subseteq \mathcal{A}$, and a pair of node types for its source and destination nodes, T_N^s and T_N^d; this is written $(:T_N^s)$-$[T_E^j{:}L^j\ \mathcal{A}^j]$->$(:T_N^d)$. Any node/edge attribute is atomic, that is, its values can only be constants. An attribute may be declared as OPTIONAL. The labels and attributes for a certain node/edge type can be specified as OPEN to indicate that nodes/edges of this type can also have other labels (resp., attributes) not explicitly specified in the schema; by default, they are not open. The graph type T_G can be specified either as STRICT (each nodes and edges must validate at least one of the corresponding specified types) or LOOSE (some nodes/edges may not comply to any type in the schema), the latter giving more flexibility.

III. DERIVING A PG SCHEMA FOR ANY DATASET

We now present our method for deriving, for any dataset (or set of datasets), a PG schema [2] starting from their abstraction, such that (i) the data conforms to the PG schema (it could be entirely converted into a PG graph valid wrt the schema), and (ii) the PG schema is relatively "tight", i.e., only datasets structurally similar to the input one would be valid wrt the schema. With this goal in mind, our target PG Schema is not OPEN.

Beyond the entities $\mathcal{E}$ and relationships $\mathcal{R}$, our algorithm (Algo. 1) takes a parameter $\phi \in \{$FLAT, CUT$\}$, specifying how to map possible nested ABSTRA attributes into PG schema node/edge attributes. Intuitively, we can either (i) "wrap" all

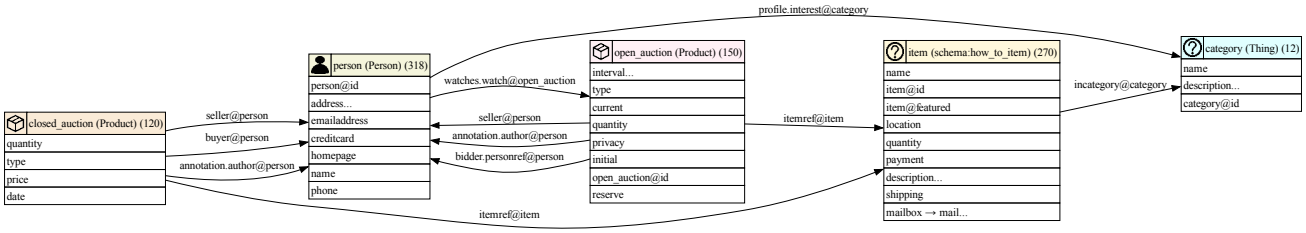

Fig. 1. Abstraction computed out of an XMark dataset (from [5]).

the content of the nested attribute into a single atomic one. At the *data* level (not discussed in this paper where we only synthesize a PG schema), this corresponds to a traversal (and serialization) of the nested ABSTRA entity attribute value, into a single field, that will be the value of the PG schema attribute (FLAT); or (ii) "cut (separate)" the nodes in the nested entity attribute, into as many standalone PG node types as needed (CUT).

First, Algo. 1 identifies a set of **PG node types**. For each entity e in $\mathcal{E}$, we compute: a node type T_N^e to which we associate a set of labels L^e and a set of attributes $\mathcal{A}^e$ (Sec. II-C). The node type and node label(s) are already provided by ABSTRA: they correspond to the entity name (the natural common collection name), respectively its semantic class (Sec. II-B). In our case, $|L^e|$=1 because ABSTRA assigns only one semantic class to an entity. For instance, in Fig. 1, the light blue entity leads to T_N^e=`item` and L^e=`Product`. Next, to build the attribute set $\mathcal{A}^e$, we iterate over each attribute $a \in e$, and proceed as follows (Lines 7-13): (i) if a has atomic values, we simply add it to $\mathcal{A}^e$, declaring it of type string; (ii) otherwise, we decide based on ϕ. If ϕ=FLAT, the attribute, with all its descendants that are still in the boundary of the main entity, are wrapped in an atomic value (Line 11). In Fig. 1, the `item`'s attribute `description` would lead to a JSON object containing `text`, `ul` and `li` (the description attributes; hidden in Fig. 1). Otherwise (ϕ=CUT), the attribute is unfolded: for each of its child attributes, a (new) node type is generated, as well as the corresponding "parent-child" edge types are created (Line 13). Using CUT, the item `description` attribute would lead to a new PG node type, having attributes `text`, `ul` and `li`; a PG edge type would connect `item` to `description`. Further, a is marked as OPTIONAL when not all records have it (Line 15). For instance, only few items have a `shipping` element, thus it is marked as optional.

Second, we compute **PG edge types**. For each pair of ABSTRA entities e_i and e_j connected by a l-labeled relationship, we get the PG node types of the source and target entities (T_N^i and T_N^j) and add to the PG schema an edge type T_E^z labeled with the ABSTRA relationship label l. For instance, in Fig. 1, the entity `person` is connected to the entity `open_auction` with a relationship labeled `watches.watchopen_auction`. Thus, in the PG schema, the corresponding PG node types `personType` and `open_auctionType` are connected by a PG edge type `Edge3Type`, labeled `Watches_watchOpen_auction` (Fig. 2).

For what concerns the **PG graph type**, if the abstraction represents 100% of the data (recall Sec. II-B), we declare the schema to be STRICT. Otherwise (if abstraction left some data out because of a limit on the size of $|\mathcal{E}|$), the resulting PG schema is LOOSE. This is because the unrepresented nodes, respectively, edges, will not comply with any of the corresponding types defined in the PG schema.

Fig. 2 shows part of the PG schema obtained from the XMark abstraction (Fig. 1) with ϕ = FLAT. The first node type `personType` comes from the abstraction itself (entity `person`), while the `addressType` comes from a nested attribute flattened as a String. An edge type is defined to connect the `personType` to the `addressType`. More edge types (`personType` to `categoryType` and `personType` to `open_auctionType`) have been declared, following ABSTRA relationships.

IV. EVALUATION

We implemented the PG schema generation algorithm in Python. Its starting point is an ABSTRA-computed abstraction, as well as the simple data graph, stored in a Postgres database.

A. Datasets

We tested our PG schema generation approach on several datasets (Tab. I), of different data models. The Companies dataset (CSV) describes the 40 most influential French companies by their id, name and Wikipedia headline; Conferences (RDF) is about scientific publications (having a title and year) and their authors (identified by their first and last names and affiliation); the JSON Researchers dataset describes authors (id, first and last names, gender, age, status) and their top-5 publications as well as their 3 most frequent 3 co-authors. Finally, XML evaluation datasets comprise: an XMark [24] dataset (Fig. 1); the HATVP dataset [14], a French public transparency dataset about elected officials' wealth; PubMed one is a sample of bibliographic notices available in PubMed, a repository of scientific biomedical literature. Double arrows (⇕) indicate datasets including entities with nested attributes, while real-life datasets are denoted by a •.

Algorithm 1: ABSTRA abstraction to PG schema

Input: N, E, $\mathcal{E}$, $\mathcal{R}$, $\phi \in \{\mathsf{FLAT}, \mathsf{CUT}\}$
Output: PG schema T_G

$T_N \leftarrow \emptyset$, $T_E \leftarrow \emptyset$;
for $e \in \mathcal{E}$ **do**
  $T_N^e \leftarrow e.name$;
  $L^e \leftarrow e.semantic\_class$;
  $\mathcal{A}^e \leftarrow \emptyset$;
  for *attribute* $a \in e.boundary$ **do**
    if *a is not a nested attribute* **then**
      $A \leftarrow a$ STRING;
    else
      if ϕ=FLAT **then**
        $A \leftarrow wrap(a)$;
      else
        $A \leftarrow unfold(a)$;
    if *all nodes in the collection e do not have a* **then**
      $A \leftarrow$ OPTIONAL A;
    $\mathcal{A}^e \leftarrow \mathcal{A}^e \cup \{A\}$;
  $T_N \leftarrow T_N \cup \{(T_N^e : L^e\ \mathcal{A}^e)\}$;
$z \leftarrow 1$;
for $e_i \xrightarrow{l} e_j \in \mathcal{R}$ **do**
  $T_E \leftarrow T_E \cup (:T_N^i)\text{-}[T_E^z{:}\ l]\text{->}\ (:T_N^j)$;
  $z \leftarrow z + 1$;
if *$\mathcal{E}$ and $\mathcal{R}$ represent all N and E, resp.* **then**
  $T_G \leftarrow \mathsf{STRICT}(T_N, T_E)$;
else
  $T_G \leftarrow \mathsf{LOOSE}(T_N, T_E)$;
return T_G;

B. Metrics

We evaluated each generated PG schema on the following points:

(i) *Size*: How do they compare to abstractions in terms of size? (Sec. IV-C);

(ii) *Correctness*: Are the PG schemas syntactically correct? (Sec. IV-D); and

(iii) *Soundness*: Are they true to the initial abstraction? (Sec. IV-D).

We did not report on scalability as the time spent to generate the PG schemas was not significant (usually, less than a second in our experiments).

C. Data abstractions vs PG schemas size

To answer (i), Tab. I shows, for each dataset, the number of nodes and edges in the simple data graph ($|N|$ and $|E|$ in Sec. II-A); the number of ABSTRA entities and relationships ($\mathcal{E}$ and $\mathcal{R}$ in Sec. II-B); the number of nodes and edges in the PG schema with ϕ=FLAT ($|N|^F$ and $|E|^F$), respectively for ϕ=CUT ($|N|^C$ and $|E|^C$).

When a data abstraction contains only simple attributes, the resulting PG schema is of the same size, regardless of the value of ϕ. This is because no attribute leads to new (additional) node types, as in the Companies and Conferences datasets.

In contrast, when the abstraction features entities with nested attributes: (i) When ϕ=FLAT, the PG schema is of the same size as the data abstraction in terms of nodes ($|N|^F$) and edges ($|E|^F$). This is the case of all datasets with nested attributes (⇳). (ii) When ϕ=CUT, the PG schema is larger than the abstraction, both in terms of nodes and edges, because new PG node and edge types are created out of the nested attributes, as in the HATVP dataset where more than 200 of each have been created. This is because the dataset is a deep tree, where some attributes have up to 69 child attributes (themselves containing few attributes), all leading to new PG node types.

Dataset	$\lvert N\rvert$	$\lvert E\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert N\rvert^F$	$\lvert E\rvert^F$	$\lvert N\rvert^C$	$\lvert E\rvert^C$
Companies•	562	640	1	0	1	0	1	0
Conferences	120	183	2	2	2	2	2	2
Research⇳	540	610	1	0	1	0	7	6
XMark⇳	44,920	45,937	5	11	5	11	11	17
HATVP⇳•	2,515,104	2,672,021	1	0	1	0	210	208
PubMed⇳•	702	955	1	0	1	0	4	3

TABLE I
PG SCHEMA SIZES FOR EVALUATION DATASETS.

```
CREATE GRAPH TYPE xmarkGraphType STRICT {
  (personType: Person {OPTIONAL phone String, emailaddress
       String, ...}),
  (addressType: Address {OPTIONAL province String, city
       String,...}),
  ...
  (:personType)-[edge1Type: PersonAddress]->(:addressType),
  (:personType)-[edge2Type: Profile_interest]->(:
       categoryType),
  (:personType)-[edge3Type: Watches_watchOpen_auction]->(:
       open_auctionType)
}
```

Fig. 2. XMark PG schema.

D. Correctness and soundness of the generated schemas

To answer (ii), we parsed our generated PG schemas using ANTLR [29] and verified that all of them are successively accepted by the grammar outlined in [2]. To answer (iii), 3 authors compared manually the abstraction E-R diagram and the generated PG schema, and answered the following questions:

(i) *Are all* ABSTRA *entities represented in the PG schema?*;

(ii) *Do attributes belong to the right entity?*;

(iii) *Are nested attributes faithfully represented in the PG schema?*; and

(iv) *Are relationships connecting the right entities with the right label?*.

The three authors have unanimously answered "Yes" to all the questions indicating that the generated PG schemas faithfully represent the initial data abstraction, including the nested elements.

V. CONCLUSION AND FUTURE WORK

We presented an approach to derive, from any (semi)-structured dataset, a PG schema following the syntax described in [2]. Among the closest related work, the W3C defined a language for expressing relational-to-RDF mappings (R2RML [22]). There have been prior works that recommend a relational schema for a semi-structured dataset, e.g. [11], [26], [6], [8]. However, to the best of our knowledge, our work is the first to aim at mapping data from a variety of formats, into PGs. To this end, we exploit simple data graphs built by CL [1], and data abstractions of [5]. Mapping heterogeneous datasets to a well-defined schema also facilitates information

discovery and exploration [21], [17]. These mappings can also be used by mediator systems [9] where all data resides "as it is" (CSV, XML, RDF, JSON, etc.) and a middle-layer "transforms/converts/manipulates" the underlying data on-demand as a property graph for PG queries.

Our next step is to migrate the data itself into the PG format. This involves automatically data translation or mappings, inspired by previously studied schema mappings, e.g., [15], [18]. These will query the database storing the simple data graph, and produce the target PG nodes and edges. Producing a PG schema and dataset out of any structured and semi-structured dataset builds towards standardized conversion of data models and datasets, enabling better compatibility and reusability.

Acknowledgments

This work is partially funded by DIM RFSI PHD 2020-01, AI Chair SourcesSay (ANR-20-CHIA-0015-01) and European Commission under Horizon Europe Programme (ELIAS-101120237).

References

[1] 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.

[2] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Alastair Green, Jan Hidders, Bei Li, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Stefan Plantikow, Ognjen Savkovic, Michael Schmidt, Juan Sequeda, Slawek Staworko, Dominik Tomaszuk, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Dusan Zivkovic. PG-Schema: Schemas for property graphs. *Proc. ACM Manag. Data*, 1(2):198:1–198:25, 2023.

[3] Mohamed Amine Baazizi, Clément Berti, Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. Human-in-the-loop schema inference for massive JSON datasets. In *EDBT*, 2020.

[4] Mohamed Amine Baazizi, Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. Parametric schema inference for massive JSON datasets. *VLDB J.*, 28(4):497–521, 2019.

[5] Nelly Barret, Ioana Manolescu, and Prajna Upadhyay. Computing generic abstractions from application datasets. In Letizia Tanca, Qiong Luo, Giuseppe Polese, Loredana Caruccio, Xavier Oriol, and Donatella Firmani, editors, *Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28*, pages 94–107. OpenProceedings.org, 2024.

[6] Philip Bohannon, Juliana Freire, Jayant R. Haritsa, Maya Ramanath, Prasan Roy, and Jérôme Siméon. Bridging the XML relational divide with LegoDB. In *ICDE*, pages 759–761, 2003.

[7] Angela Bonifati, Stefania-Gabriela Dumbrava, Emile Martinez, Fatemeh Ghasemi, Malo Jaffré, Pacome Luton, and Thomas Pickles. DiscoPG: Property graph schema discovery and exploration. *Proc. VLDB Endow.*, 15(12):3654–3657, 2022.

[8] Mihaela A. Bornea, Julian Dolby, Anastasios Kementsietsidis, Kavitha Srinivas, Patrick Dantressangle, Octavian Udrea, and Bishwaranjan Bhattacharjee. Building an efficient RDF store over a relational database. In *SIGMOD*, pages 121–132, 2013.

[9] Maxime Buron, François Goasdoué, Ioana Manolescu, and Marie-Laure Mugnier. Ontology-based RDF integration of heterogeneous data. In Angela Bonifati, Yongluan Zhou, Marcos Antonio Vaz Salles, Alexander Böhm, Dan Olteanu, George H. L. Fletcher, Arijit Khan, and Bin Yang, editors, *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*, pages 299–310. OpenProceedings.org, 2020.

[10] Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. Schemas for safe and efficient XML processing. In Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan, editors, *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 1378–1379. IEEE Computer Society, 2011.

[11] Alin Deutsch, Mary F. Fernández, and Dan Suciu. Storing semistructured data with STORED. In *SIGMOD*, pages 431–442, 1999.

[12] François Goasdoué, Pawel Guzewicz, and Ioana Manolescu. Incremental structural summarization of RDF graphs. In Melanie Herschel, Helena Galhardas, Berthold Reinwald, Irini Fundulaki, Carsten Binnig, and Zoi Kaoudi, editors, *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019*, pages 566–569. OpenProceedings.org, 2019.

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

[14] Haute autorité pour la transparence de la vie publique. https://www.data.gouv.fr/fr/organizations/haute-autorite-pour-la-transparence-de-la-vie-publique/.

[15] Phokion G. Kolaitis, Reinhard Pichler, Emanuel Sallinger, and Vadim Savenkov. Limits of schema mappings. In Wim Martens and Thomas Zeume, editors, *19th International Conference on Database Theory, ICDT 2016, Bordeaux, France, March 15-18, 2016*, volume 48 of *LIPIcs*, pages 19:1–19:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[16] Hanâ Lbath, Angela Bonifati, and Russ Harmer. Schema inference for property graphs. In Yannis Velegrakis, Demetris Zeinalipour-Yazti, Panos K. Chrysanthis, and Francesco Guerra, editors, *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, pages 499–504. OpenProceedings.org, 2021.

[17] Lacramioara Mazilu, Norman W. Paton, Alvaro A. A. Fernandes, and Martin Koehler. Schema mapping generation in the wild. *Inf. Syst.*, 104:101904, 2022.

[18] Giansalvatore Mecca, Paolo Papotti, and Salvatore Raunich. Core schema mappings: Scalable core computations in data exchange. *Inf. Syst.*, 37(7):677–711, 2012.

[19] Neo4j Property Graphs. https://neo4j.com/product/neo4j-graph-database/.

[20] Oracle Property Graphs. https://docs.oracle.com/en/database/oracle/property-graph/.

[21] Norman W. Paton, Jiaoyan Chen, and Zhenyu Wu. Dataset discovery and exploration: A survey. *ACM Comput. Surv.*, 56(4):102:1–102:37, 2024.

[22] R2RML: RDB to RDF Mapping Language. https://www.w3.org/TR/r2rml/.

[23] Raghu Ramakhrishnan and Johannes Gehrke. *Database Management Systems (3rd edition)*. McGraw-Hill, 2003.

[24] Albrecht Schmidt, Florian Waas, Martin L. Kersten, Michael J. Carey, Ioana Manolescu, and Ralph Busse. XMark: A benchmark for XML data management. In *Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002, Hong Kong, August 20-23, 2002*, pages 974–985. Morgan Kaufmann, 2002.

[25] Juan F. Sequeda, Marcelo Arenas, and Daniel P. Miranker. On directly mapping relational databases to RDF and OWL. In Alain Mille, Fabien Gandon, Jacques Misselis, Michael Rabinovich, and Steffen Staab, editors, *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012*, pages 649–658. ACM, 2012.

[26] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *VLDB*, pages 302–314, 1999.

[27] William Spoth, Oliver A Kennedy, Ying Lu, Beda Hammerschmidt, and Zhen Hua Liu. Reducing ambiguity in JSON schema discovery. In *SIGMOD*, 2021.

[28] Radu Stoica, George H. L. Fletcher, and Juan F. Sequeda. On directly mapping relational databases to property graphs. In Aidan Hogan and Tova Milo, editors, *Proceedings of the 13th Alberto Mendelzon International Workshop on Foundations of Data Management, Asunción, Paraguay, June 3-7, 2019*, volume 2369 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019.

[29] ANTLR Lab. http://lab.antlr.org/.

[30] Offshore Leaks Database. https://offshoreleaks.icij.org/pages/database.

[31] W3C XSD. https://www.w3.org/TR/xmlschema11-1/, 2012.