User-friendly exploration of highly heterogeneous data lakes

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

User-friendly exploration of highly heterogeneous data lakes

Nelly Barret[1][0000−0002−3469−4149], Simon Ebel[1], Théo Galizzi[1], Ioana Manolescu[1][0000−0002−0425−2462], and Madhulika Mohanty[1][0009−0004−9446−8663]

Inria and Institut Polytechnique de Paris, France
firstname.lastname@inria.fr

Abstract. The proliferation of digital data sources and formats has led to the apparition of *data lakes*, systems where numerous data sources coexist, with less (or no) control and coordination among the sources, than previously practised in enterprise databases and data warehouses. While most data lakes are designed for very large number of *tables*, ConnectionLens [2,3] is a data lake system for structured, semi-structured, and unstructured data, which it integrates into a single *graph*; the graph can be explored via graph queries with keyword search [4] and entity path enumeration [5]. In this paper, we describe ConnectionStudio, a user-friendly platform leveraging ConnectionLens, and integrating feedback from non-expert users, in particular, journalists. Our main insights are: (*i*) improve and entice exploration by giving a *first global view*; (*ii*) facilitate tabular exports from the integrated graph; (*iii*) provide interactive means to improve the graph constructions. The insights can be used to further advance the exploration and usage of data lakes for non-IT users.

Keywords: Heterogeneous data · Data lake · Data exploration.

1 Outline: highly heterogeneous data lakes

The past few decades have seen an important rise in the production of digital data. This data spans across multiple domains, e.g., healthcare, environment, finance, administration; it is owned and used by many actors other than the data producers, notably data journalists and researchers for crucial data journalism applications. The heterogeneity poses multiple challenges for data integration, its exploration and understanding of the data. **Data lakes** [9,7,12,10] are centralized repositories designed to store, process, and secure large amounts of structured, semi-structured, and unstructured data. A data lake stores data in its native format and supports various styles of data processing; the data model most often considered in such settings is relational, e.g., [8,6,11]. ConnectionLens [2,3] is a data lake system integrating such heterogeneous data in a *graph* format, capturing the fine-granularity structure that (semi-)structured data sources may have. Further, ConnectionLens applies Information Extraction techniques

to identify, from any value (leaf) node encountered in the data, entities such as people, places, organizations, emails, URIs, dates, etc. Such entities are very appealing in particular to data journalists, because they are at the heart of their work: analyzing the activity of entities of particular interest, e.g., political leaders, or companies, and finding how those entities may be connected. ConnectionStudio supports querying this integrated graph using keyword search [4] and entity path enumeration [5]. Using keyword search, users submit keywords that interest them, and ConnectionLens returns *connecting trees*, showing how, in the graph, nodes matching the keywords are connected. When the users know the types of entities of interest (which is the case with data journalists), an efficient *entity path enumeration algorithm* enumerates and allows visualizing paths that connect entities of interest, e.g., a politician owns shares in a company, or a politician's wife serves in the governing board of a company, etc.

New requirements for non-expert users Working with journalists, we found that heterogeneous graphs produced by ConnectionLens were still hard for them to comprehend. They found it difficult (α) relating the documents they added to the data lake, to the resulting data graphs; (β) figuring out what keywords to use when searching; (γ) generally, working with the data graph paradigm. It turns out that in their profession, those with digital skills are especially at ease with spreadsheet tools, thus data tables are appealing to them, while data graphs are not. Further, (δ) ConnectionLens graphs interconnect data through extracted entities; like any trained model, our extractors introduce some errors (false positives, false negatives, wrongly typed entities). In some cases, especially when a dataset has some regularity, users can provide guidance on what entities are to be expected in certain parts of the data, thus contribute to increasing the graph quality. To articulate such guidance, they need to be able to inspect the data, and to formulate extraction hints. Moreover, (ϵ) journalists formulated the need for tangible, intuitive data analysis results (diagrams, graphs, tables) that they can download from our graph data lake and share, for instance, within newsrooms, to convince colleagues or managers of the interest of spending time to analyze complex data.

Based on these requirements, we built **ConnectionStudio**, a new platform based on ConnectionLens and extending it in several ways in order to address (α) to (ϵ) above. Below, after recalling basic information about ConnectionLens to make this paper self-contained, we detail ConnectionStudio's novel extensions, which are the contributions of this work. We believe they may help others devising similar heterogeneous data lakes. ConnectionStudio is available online, together with examples and tutorials at `https://connectionstudio.inria.fr/`.

2 Background: graph integration of heterogeneous data

ConnectionLens ingests any structured, semi-structured or unstructured data as follows. When ingesting an XML document, each element, attribute, or text node becomes a graph node; parent-child relationships in the XML document lead to

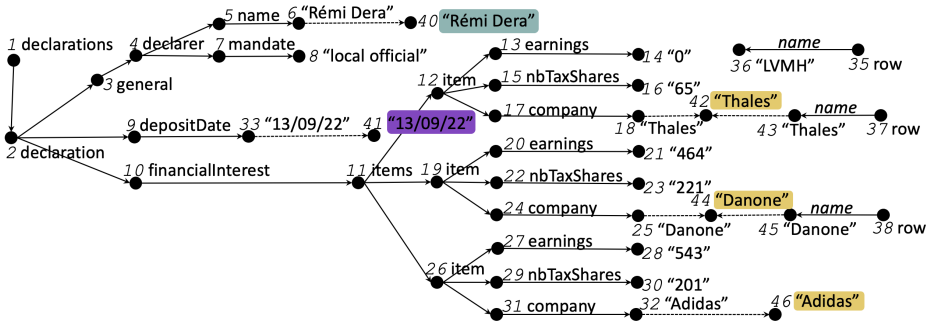


Fig. 1. Sample data graph built from HATVP declarations and CAC40 companies.

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 paragraph, 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 connected to the leaf value from which it has been extract through an edge (dashed edges in Figure 1). Moreover, when two NE nodes are identical, i.e. they have the exact same label, they are fused and only one is kept in the integrated graph. This allows to easily find connections *across sources*, that are (much) harder to find manually. For instance, there is only one node "Thales", extracted from N18 and N43.

Figure 1 shows the data graph obtained from (*i*) an XML sample of the large HATVP French transparency dataset (ministers' declarations about their wealth, stocks they own, business interests, etc.), on the left; and (*ii*) a CSV file listing the 40 most influential French companies (known as CAC40), on the right. Each (black) circle is a data node in the integrated data graph, edges are connecting them accordingly to their relationships in the datasets. Value data nodes are quoted; named entities are highlighted (blue for people, purple for dates and yellow for organizations).

3 Novel ConnectionStudio modules

Built on top of ConnectionLens, ConnectionStudio allows users to import datasets into an integrated graph, search the graph via keywords, and find paths connecting entities. Further, ConnectionStudio includes new modules, to answer the requirements stated at the end of Section 1. We describe them below.

3.1 Global view of the data lake: entity (dataset) statistics

Users are familiar with the data files they brought (PDF, Office formats, JSON, CSV, etc.), but told us they felt "lost" once the system ingested their data, especially if the latter is large and/or complex. To help them get a first global view of the data lake (or graph, requirement (α)), we present them a set of *entity* and *entity-dataset statistics*, as follows:

- The total numbers of entities of each type (Person, Location, Organization, date, URI, email, hashtag, mention), overall in the graph;
- The total number of entities per type and per dataset in which they appear;
- A tag cloud of the most frequent entities in the whole graph.
- A summary of the *entity-dataset associations*: we show the entity type, label, and datasets where it appears, starting with the entities present in the highest number of datasets. These entities are more interesting, because they allow making connections *across datasets* (potentially heterogeneous files), saving important manual efforts to journalists combining such data sources for their investigations.

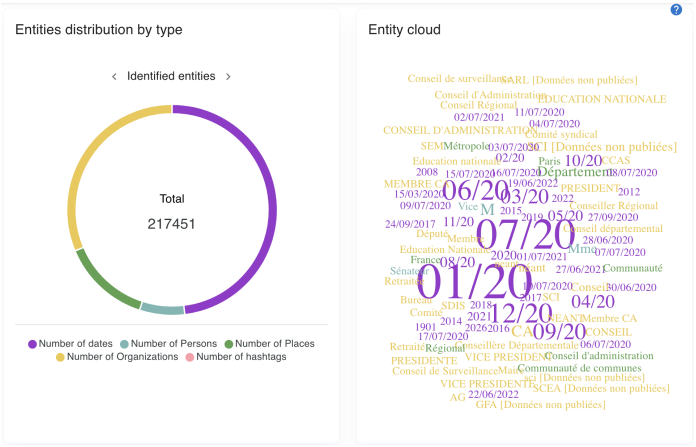
The above statistics give a first idea of what the datasets contain, and also suggest entity names to use as search keywords (requirement (β)). For instance, Figure 2a shows the number of different types of extracted entities and the tag cloud in the dataset of Figure 1. Figure 2b shows the frequent common entities in the two datasets.

3.2 From the integrated graph to data tables

In a prior ConnectionLens application [3], based on PubMed biomedical literature data, journalists were interested in the paths that connect medical experts with companies that fund their research. We expressed this path as a query over the graph, and exposed its results as a table through an ad-hoc Web interface manually built just for this scenario. In a more general manner, in [5], users select a pair of entity types (τ_1, τ_2) of interest to them, e.g., people and organizations, and the system automatically finds and computes the paths connecting such entities in the ConnectionLens graph. Each path leads to a data table, where entities of types τ_1, τ_2 are in the first, resp., last column, and the other columns are the nodes along the path.

As stated in requirement (γ), *users requested more support in extracting tabular data from the graph*. For instance, in the HATVP dataset, they may want to extract: for each elected politician, their name, elected office, election date, and companies they have stocks in; note that this query is not a path, since it returns many values. Users view the first three fields as required, but the last one is optional, i.e., a politician should be part of the result even if they have no stocks. Optional query fragments correspond to *outerjoins* in database terms.

To allow users to *easily and intuitively express a much larger set of queries*, we proceed as follows.



(a) Count of entities and tag cloud

COLUMNS FILTERS DENSITY EXPORT

Label	Type	Freq…	Datasets
2017	Date	452	Cac40.csv, hatvp-cleaned.xml
Legrand	Organization	7	Cac40.csv, hatvp-cleaned.xml
Engie	Organization	20	Cac40.csv, hatvp-cleaned.xml
2000	Date	63	Cac40.csv, hatvp-cleaned.xml
2009	Date	29	Cac40.csv, hatvp-cleaned.xml
Pernod Ricard	Organization	6	Cac40.csv, hatvp-cleaned.xml
Alsace	Location	8	Cac40.csv, hatvp-cleaned.xml

(b) Frequent entities

Fig. 2. Statistics for the sample dataset.

(1) Upon loading, ConnectionStudio computes, from each dataset, a set of *elementary paths* that can be seen as "query building blocks". Each path reflects one or more consecutive edges in the data graph. The source of a path is always an internal node, while its destination is either an internal node, a value, or an extracted entity. For instance, in the above example, elementary paths include: `declarations`, `declarations.declaration`, `declarations.declaration.general.declarer.name#val` (this ends in strings comprising politicians' names), `declarations.declaration.general.declarer.name#val.extract:p` (ending in person entities extracted from the strings), etc.

(2) Users can select paths from a drop-down list, and add them one by one to compose a query. The first selected path is required; the others can be either optional or required. To each path are associated two variables: one for the source node, and the other for the destination (internal node, value, or entity). Users

can edit the paths, and the variables, to adjust them, and specify how they connect. For instance, in the HATVP scenario, a user may:

- Start by selecting a path ending in `declaration`; name its starting point (source variable) `decls` and its end point (target variable) `decl`.
- Select `declarations.declaration.general.mandat.label#val` as the second elementary path;
- Edit (shorten) it into `declaration.general.mandat.label#val`, going from the variable `decl` to the variable `position`. Reusing `decl` is intuitive since in both paths, this variable denotes the graph nodes labeled `declaration`.
- Similarly, edit other elementary paths to obtain `declaration.general.dateDebutMandat#val.extract:d` going from `decl` to `startDate`, etc.

(3) When the user has finished specifying the paths to combine in a query, they can trigger the evaluation of this query on the underlying graph. This leads to tabular results, with a column for each user-specified variable and a line for each result; users can download results in CSV to be further processed, shared etc.

For instance, Figure 3 shows the result of joining three paths and renaming the variables to obtain the declaration number, the start date, the position and the name of the person.

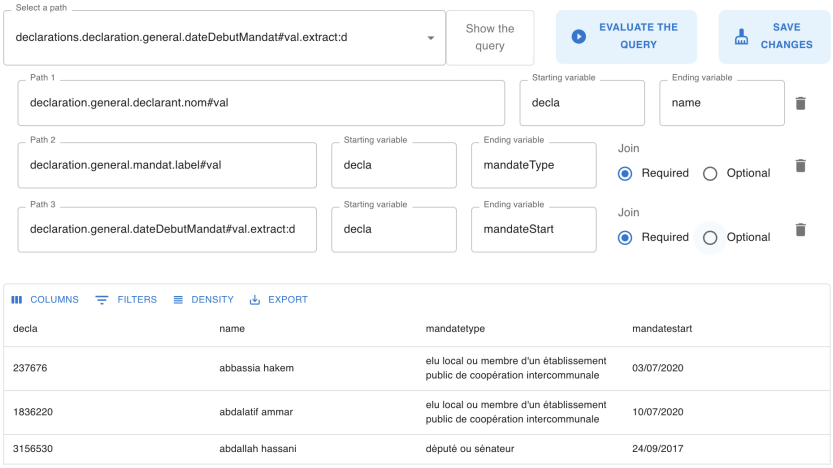


Fig. 3. The data view for HATVP declarations and CAC40 companies samples.

Generating elementary paths As explained above, users can compose queries by "cutting & pasting" elementary paths; here is how ConnectionStudio extracts these from the data. From an XML or JSON document, each path starting from the document root, and ending in an internal node, text node, or extracted entity (child of a text node) is proposed to the user. From CSV data, we propose

paths of the form `row`, `row.att#val`, `row.att#val.extract:`τ where `row` is the label of each node created out of a CSV row (tuple), `att` is an attribute name, and `extract:`τ denotes an extraction edge for some entity type τ (such as person, location, email etc., recall Section 2). From RDF, for each property `p` encountered in an `s, p, o` triple, we propose simply `p` as an elementary path, with two variables for the subject and object of the triple; similarly, for each `s, rdf:type, c` triple, we propose `rdf:type c` as an elementary path with one variable for the subject.

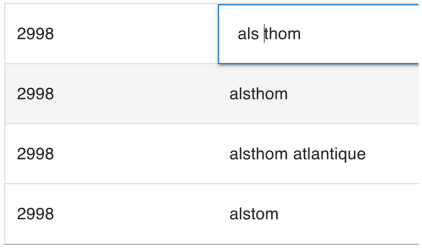
3.3 Correcting and improving the graph through a table view

The paradigm of path querying also gives us two ways to improve and correct the graph (requirement (δ)).

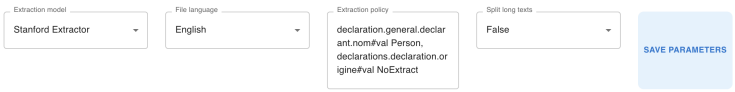
Editing value and entity labels As stated above, an elementary path ending in `#val` returns the set of values encountered in the data in certain positions, while a path ending in `#val.extract:`τ, for some entity type τ, shows entities extracted by ConnectionLens from the data, using trained language models [3]. When visualising the result of such a query, users can *edit* entity or values shown in the query result, and *propagate* their modifications to the underlying database, thus updating the graph. ConnectionLens implements a set of similarity functions and (very conservatively) unifies entities whose labels are very similar, e.g., "L'Oréal" and "L'OREAL", once they are both recognized as organizations by the entity extractor. The ability to edit the data, offered by ConnectionStudio enables users to further normalize (uniformize) the label of value nodes, and/or of extracted entity nodes. This corresponds to a carefully restricted case of *database update through views* [1]. As well known, such updates cannot always be propagated correctly to the underlying database. In ConnectionStudio, we allow updates only on values or entities at the destination end of a path. It is easy to see that such updates can always be propagated to the graph persistently stored in the underlying database.

For instance, in Figure 4a, while inspecting the results of a query, when users find multiple versions of the same organization "Alstom", such as "Alsthom", "Als thom" or "Alstom grid", they can correct each of them by hand, then propagate the changes to the underlying database.

Specifying extraction policies Inspecting results of entity-returning path queries may help users learn what entities are (not) in specific places in the data. Thus, a user noticing the extracted names "Bertrand Martin" and "Julie Dupont" under `declaration.general.declarer.name#val.extract:p` may conclude that every `declaration.general.declarer.name#val` contains people, and formulate an *extraction policy* of the form $\boxed{path\ \tau}$, specifying that all values found under *path* are to be interpreted as entities of the given entity type τ. This helps circumvent extractor misses, e.g., for a less usual name such as "Xin Jong" which does not fit the extractor's trained model. Users can also specify that extraction should *not* be performed on values on some path(s), $\boxed{path\ NoExtract}$, if they are not inter-



(a) Updating values to clean the data.



(b) Specify extraction policies before loading data.

Fig. 4. Correcting and improving the graph.

ested in the entities that may be found there. Extraction policies, both negative and positive, speed up the graph construction, by avoiding the (costly) entity extraction effort during graph loading.

Extraction policies were first mentioned in [3]. However, only now, via ConnectionStudio's path query features, our tool helps non-expert users formulate them. For instance, in Figure 4b, the user specified that: declarers' names should always be recognized as person entities, and that no extraction should be applied on the values found on the path `origine#val`. Users can decide this after seeing that all these values are equal (probably a code introduced by anonymization), thus there is no point in searching for entities in them.

4 Perspectives and conclusion

Data lakes such as [9,7,12,10] and ConnectionLens [2,3] aim to help users explore many heterogeneous data sources. ConnectionLens adopts a graph paradigm for integrating the sources, and extracts entities leading to inter-dataset connection opportunities. In this work, we describe novel data exploration and discovery paradigms we implemented in ConnectionStudio, following requirements expressed by journalists; they allow users to *discover* the graph, *simplify querying* for connections across sources, and *as-you-go* cleaning of the graph. We believe these features are useful additions to next-generation heterogeneous data lakes. Going forward, we plan to conduct an elaborate user-study in order to understand better how ConnectionStudio helps novice users explore graphs and also inculcate the feedback to further improve upon the features provided by ConnectionStudio.

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.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of databases. Addison-Wesley (1995)
2. Anadiotis, A., Balalau, O., Bouganim, T., et al.: Empowering investigative journalism with graph-based heterogeneous data management. IEEE DEBull. (2021)
3. Anadiotis, A., Balalau, O., Conceicao, C., et al.: Graph integration of structured, semistructured and unstructured data for data journalism. Inf. Systems **104** (2022)
4. Anadiotis, A.C., Manolescu, I., Mohanty, M.: Integrating Connection Search in Graph Queries. In: ICDE (Apr 2023)
5. Barret, N., Gauquier, A., Law, J.J., Manolescu, I.: Exploring heterogeneous data graphs through their entity paths. In: ADBIS (2023)
6. Fan, G., Wang, J., Li, Y., Zhang, D., Miller, R.J.: Semantics-aware dataset discovery from data lakes with contextualized column-based representation learning. PVLDB **16**(7) (2023)
7. Giebler, C., Gröger, C., Hoos, E., Schwarz, H., Mitschang, B.: Leveraging the data lake: Current state and challenges. In: Big Data Analytics and Knowledge Discovery (DaWaK). Springer (2019)
8. Giebler, C., Gröger, C., Hoos, E., Schwarz, H., Mitschang, B.: Modeling data lakes with data vault: Practical experiences, assessment, and lessons learned. In: ER. Lecture Notes in Computer Science, vol. 11788. Springer (2019)
9. Hai, R., Geisler, S., Quix, C.: Constance: An intelligent data lake system. In: SIGMOD. New York, NY, USA (2016)
10. Hai, R., Koutras, C., Quix, C., Jarke, M.: Data lakes: A survey of functions and systems. IEEE Transactions on Knowledge and Data Engineering (2023)
11. Kuschewski, M., Sauerwein, D., Alhomssi, A., Leis, V.: BtrBlocks: Efficient columnar compression for data lakes. Proc. ACM Manag. Data **1**(2) (2023)
12. Nargesian, F., Zhu, E., Miller, R.J., Pu, K.Q., Arocena, P.C.: Data lake management: challenges and opportunities. Proceedings of the VLDB Endowment **12**(12) (2019)