# ConnectionStudio: User-friendly Exploration of Highly Heterogeneous Data Lakes

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

# ConnectionStudio: User-friendly Exploration of Highly Heterogeneous Data Lakes

Nelly Barret, Nikola Dobričić, Simon Ebel, Théo Galizzi, Ioana Manolescu, Madhulika Mohanty
Inria and IPP
France
firstname.lastname@inria.fr

## ABSTRACT

To handle increasingly large and varied sets of digital data sources, *data lakes* have been gaining popularity as modern data integration platforms. In contrast with mediators and data warehouses, data lakes do not attempt to establish a single common schema. While most data lakes are designed for large collections of *tables*, ConnectionLens [2, 3] is a data lake system which uniformly ingests structured, semi-structured, and unstructured data into a *directed graph* that copies exactly the original datasets' structure; in a ConnectionLens instance, named entities identified though information extraction in any of the datasets can link them. Connections between nodes in a ConnectionLens datalake can be sought (*i*) through keyword search [4], or (*ii*) asking for connections between named entity pairs of specific types [6].

We propose to demonstrate ConnectionStudio, a user-friendly platform recently outlined in [5], which we built to enable non-expert users, in particular journalists, to explore ConnectionLens data lakes. The novelties of ConnectionStudio are: (*i*) improving and enticing exploration by giving a *first global view*; (*ii*) facilitating tabular exports from the integrated graph; (*iii*) interactive means to improve dataset ingestion in the lake.

## KEYWORDS

Heterogeneous data, Data lake, Data exploration.

## 1 OUTLINE: HIGHLY HETEROGENEOUS DATA LAKES

There is a rise in the production of digital data that spans across multiple domains, e.g., healthcare, environment, finance, administration. Published data has many potential users (or consumers), including data journalists and researchers for crucial data journalism applications. Data heterogeneity poses multiple challenges for data integration, exploration and understanding. ConnectionLens [2, 3] is a data lake system for heterogeneous data. It copies (converts) any incoming dataset into a *directed graph*, which preserves any structure that it may have had (tuples, nodes, edges, etc.), ensuring the possibility to restore it identically in its original format if needed. Further, ConnectionLens applies Information Extraction techniques over any value (leaf) node encountered in the data, to identify entities such as people, organizations, 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 they may be connected. ConnectionLens supports finding information in the data via two methods: (*i*) through keyword search [4], (*ii*) by enumerating all labeled paths connecting pairs of entities of user-specified types [6].

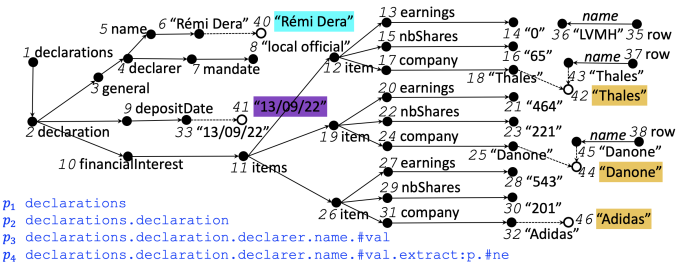


**Figure 1: Sample data graph built from HATVP declarations and CAC40 companies.**

Showing ConnectionLens to journalists, we found that understanding and querying the heterogeneous graphs produced by ConnectionLens are still hard for them. Their main requirements are: (**R1**) Relate the added documents to the resulting data graphs; (**R2**) Find keywords to use when searching; (**R3**) Download tangible, intuitive data analysis results (diagrams, graphs, tables) from the lake to be shared, e.g., within newsrooms; (**R4**) Inspect and provide extraction hints when the trained model goes wrong; (**R5**) A guided interface for expressing queries over ConnectionLens graphs.

Based on these requirements, we built **ConnectionStudio**, a new platform to help build, query and explore a ConnectionLens data lake. ConnectionStudio extends metadata in several ways, in order to address (**R1**) to (**R5**) 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.

## 2 BACKGROUND: CONNECTIONLENS DATA LAKE FOR HETEROGENEOUS DATA

Data lakes [9, 11, 12, 14] are centralized repositories designed to store, process, and analyze large amounts of data in their native format. The data model most often considered in such settings is relational, e.g., [8, 10, 13]. To help users make sense of the data without a relational schema, data lakes extract a rich variety of metadata [12], such as structural summaries, semantic annotations, data provenance, etc. This allows users to grasp the content of their datasets, but also their possible inter-connections. The ConnectionLens data lake ingests structured, semi-structured, and unstructured data into a graph $G = (N, E)$ where $N$ is a set of nodes and $E = N \times N$ is a set of directed edges. Each node and edge has a label from a set of labels $\mathcal{L}$ (including the empty label $\epsilon$). ConnectionLens data ingestion *preserves all the structure from the original dataset*, e.g., relationships between nodes and their children, tuples and their attributes, etc. Each XML element, attribute, or text node

becomes a graph node; parent-child relationships lead to $\epsilon$-labelled edges. 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 a 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 text 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 extracted through an edge. 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. Recognized entity types include: Person, Location, Organization, date, URI, email, hashtag and mention.

## 3 HELPING USERS EXPLORE THE GRAPH DATA LAKE

We now describe the novel features that ConnectionStudio adds on top of the ConnectionLens data lake, answering the journalists' requirements identified in Sec. 1. First, ConnectionStudio computes and shows to users lake-wise statistics (Sec. 3.1); these can be seen as a form of metadata, typically computed in data lakes to help users understand and work with the data despite the lack of a single schema. Second, we provide a simple, intuitive query interface, based on drop-down menus, enabling users to form simple or complex queries *within and across datasets* from the graph lake (Sec. 3.2). This is a crucial feature for investigative journalism, which often needs to find connections made through people, organizations, etc., across multiple data sources. Last but not least, ConnectionStudio provides an interface enabling users to inspect and correct errors in the original data and/or in extraction results (Sec. 3.3).

**Dataset: the French political and economical sphere** For illustration, we rely on a data lake for investigating French politicians' interests in French companies. Fig. 1 illustrates a data graph obtained from (*i*) the HATVP French transparency dataset (ministers' declarations about their wealth, stocks they own, business interests, etc.), in XML, on the left; and (*ii*) a CSV file with the 40 most valuable French companies (known as CAC40), on the right. Each (black) circle is a data node in the integrated data graph; edges connect nodes according to their relationships in the datasets (recall Sec. 2). Value data nodes are quoted; named entities are highlighted (blue for people, purple for dates and yellow for organizations) and connected to their leaf value with dashed edges. Note the connections between the datasets, e.g., through the node "Thales", extracted from N18 and N43.

### 3.1 Metadata computation

Journalists told us they felt "lost" once the system ingested their files (PDF, XLS, JSON, etc.), not being sure what the data looks like, what it contains, and what connections can be made across data sources. To help them, ConnectionStudio computes a set of **metadata**: statistics, dataset abstractions, and elementary paths.

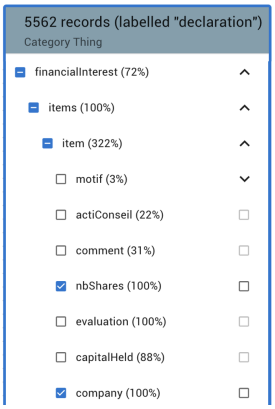


**Figure 2: Querying the data lake through data abstractions.**

**Entity (dataset) statistics** To bring users a first global view of the data lake (requirement (R1)), ConnectionStudio presents them a set of *entity* and *entity-dataset statistics*, as follows: (*i*) the total numbers of entities of each type (recall Sec. 2); (*ii*) the total number of entities per type and per dataset; (*iii*) a tag cloud of the most frequent entities in the data lake; and (*iv*) a summary of the *entity-dataset associations*, for which we show the entity type, label, and datasets where it appears, starting with the entities present in the highest number of datasets.

The above statistics give a preliminary idea of what the datasets contain, and also suggest entity names to use as search keywords (requirement (R2)). These entities are already interesting, because they allow making connections *across datasets* (potentially heterogeneous files), saving important manual efforts to journalists combining such data sources for their investigations.

**Dataset abstractions** ConnectionStudio computes out of each dataset, regardless of its data model, an *abstraction* [7], or structural summary in the style of an Entity-Relationship diagram. Unlike classical E-R diagrams, the abstractions computed from semi-structured data may have *attributes with nested internal structure*, reflecting the imbrication in complex values. For instance, the abstraction of the XML dataset in our scenario has a single entity of `declaration` elements, with attributes including `financialInterest`, having a unique (sub-)attribute `items`, itself having a (sub-)attribute `item`, which has three (sub-)attributes: `earnings`, `nbShares` and `company`. Depending on the dataset complexity, the abstraction may have more or less entities; users can bound the number $k$ of entities in the abstraction, and they will see the entities containing the most data from the original dataset. Journalists confirmed that E-R style, graphical abstractions are very helpful for grasping semi-structured dataset organization, as shown in Fig. 2. Abstractions are also one support used in ConnectionStudio to help users extract custom datasets from the ConnectionLens lake (Sec. 3.2).

**Elementary paths** Upon loading, ConnectionStudio computes, from each dataset, a set of *elementary paths* reflecting the dataset

structures. These will allow users to query the data lake (Sec. 3.2) and improve the graph quality (Sec. 3.3).

Each elementary path $p$ is a sequence of alternating node and edge labels $p = (l_{n_1}.l_{e_1}.l_{n_2}.l_{e_2}.\ldots.l_{n_k})$ for some $k \geq 1$, s.t. $n_i \in N$, $e_j \in E$ having the labels $l_{n_i}$ and $l_{e_j}$ respectively. 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 strings of politician names; `declarations.declaration.declarer.name.#val.extract:p.#ne` ends in Person entities extracted from these strings (recall Sec. 2); note the special extraction edge label `extract:p`. Paths are shown to users somehow simplified (without loss of information), e.g., we show `declarations.declaration`, not `declarations.`$\epsilon$`.declaration`. From XML or JSON documents, ConnectionStudio extracts each path starting from the document root, and ending in an internal, text, or extracted entity node. From relational data, we extract each path starting in a tuple node and ending in a value or entity node. From RDF, for each property `p` encountered in an ⟨s⟩⟨p⟩⟨o⟩ triple, we extract simply `p` (formally $\epsilon$`.p.`$\epsilon$) as an elementary path; similarly, for each ⟨s⟩⟨`rdf:type`⟩⟨c⟩ triple, we propose $\epsilon$`.rdf:type.c`).

## 3.2 Querying the data lake

ConnectionStudio provides intuitive, mostly syntax-free means to express complex queries that are composed and evaluated by the database holding the ConnectionLens data lake (in the current implementation, via Postgres), thus addressing requirement (R5).

**Querying based on abstractions** When shown a graphical 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. 2. When they are done, ConnectionStudio automatically builds a graph pattern query which extracts from the data graph the selected entity and/or relationship attributes. The query is evaluated, and the results are shown as a table. Such queries are very helpful to extract data from each dataset (requirement (R1)), however, they are restricted to one dataset at a time. Also, if users have limited the number of entities in the abstractions, the unseen entities will not be accessible to queries.

**Querying with elementary paths** Separately, in the "Data View" interface, ConnectionStudio allows users to easily form queries over several datasets as follows (Fig. 3). Choosing one dataset leads to a drop-down menu of the dataset's elementary paths. The user selects one or several paths. The first is required; the others may be required or optional. Each path is attached a "start" and an "end" variables, whose names are set by the user; sharing a variable name across two paths 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 extracted entities, such queries may express *arbitrary structural patterns* (joining on nodes), as well as *value- or entity-based joins across datasets* (and possibly different data models). It is also possible to use the same path more than once, leading to self-joins, etc. Evaluating $q$ leads to tabular results, which can be downloaded, thus addressing requirement (R3). For instance, in our scenario, journalists 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. 3) by joining four XML elementary paths: $p_1$ which returns the French officials' names, $p_2$ their declared financial interests (`item` elements), $p_3$ the companies in which they have stocks, and $p_4$ the number of their shares in the company. To specify that we want the results only for CAC40 companies, we add a fifth path (the last in Fig. 3) coming from the CSV dataset. This path is joined with the previous ones, on a common variable with $p_3$, namely `companyName`.

As an *advanced* feature, we show users the query generated from their selected path, and allow them to edit it. This is how they can for instance limit the results, add aggregation, etc.

## 3.3 Semi-supervised cleaning

Using the elementary paths, ConnectionStudio supports two ways to improve and correct the graph (requirement (R4)), a necessary feature to build a high quality data lake.

**Editing value and entity labels** As stated before, an elementary path ending in `#val` returns the set of values encountered in the data in certain positions; a path ending in `#val.extract:`$\tau$`.#ne`, for some entity type $\tau$, shows entities extracted by ConnectionLens from the data, using trained language models [3]. When visualising the result of such elementary path queries (Sec. 3.2), users can *edit* entities or values shown in the query result, and *propagate* their modifications to the underlying graph, thus updating the lake. 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 named entity extractor. The ability to thus edit the data 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, not all such updates can be propagated correctly to the underlying database. ConnectionStudio allows updating only values or entities at the destination end of a path; such updates can always be propagated to the graph.

**Specifying extraction policies** Inspecting results of elementary path queries ending in extracted entities helps users learn what entities are (not) in specific places in the data. Thus, a user noticing people names such as "Alain Rousset" and "Edouard Courtial" under `...declarer.name.#val` may conclude that every declarer name contains people, and formulate an *extraction policy* of the form $\boxed{path\ \tau}$, specifying that all values found under this path should be interpreted as entities of the given entity type $\tau$, *without* running the trained model on the values. Users can also specify an extraction policy of the form $\boxed{path\ NoExtract}$, if they deem that entities in some text values are not worth extracting. On one hand, this reduces graph construction time, dominated by Information Extraction [3]. On the other hand, a policy of the form $\boxed{path\ \tau}$

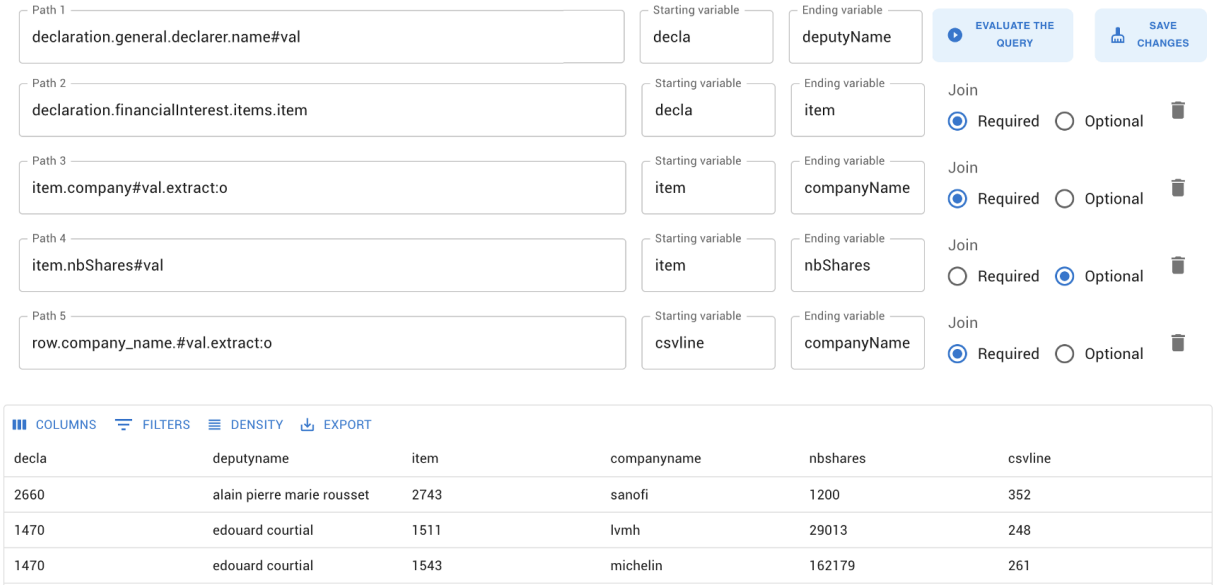


**Figure 3: Querying the data lake using elementary paths to know more about politicians' tax shares.**

helps reduce extractor misses and errors, inevitable when using a trained AI model. Extraction policies were first mentioned in [3] as an help for entity extraction. They now also serve as an intuitive way to query data.

## 4 DEMONSTRATION SCENARIO

ConnectionStudio is developed in Java 11. It acts as a front-end to ConnectionLens [3], includes abstraction computation [7] and enumeration of paths between named entities [6], as well as the novel features (Sec. 3). The data graph and all metadata are stored in PostgreSQL, thus graph queries are translated to SQL. We will demonstrate ConnectionStudio on real-life datasets. Demo attendees will be able to ingest, inspect, query, and modify a data lake describing how French politicians relate to CAC40 companies, as in our running scenario. A second data lake we prepared contains tweets from French politicians (JSON) together with an RDF graph, extracted from Wikidata, describing the main French political parties. The third data lake features PubMed XML data (bibliographic notices of biomedical papers), together with media articles (HTML) about pharmaceutical companies; this dataset clearly needs more normalization, especially for company names. This will enable experimenting with semi-supervised cleaning and entity extraction policies. ConnectionStudio is available online, together with examples and tutorials at https://connectionstudio.inria.fr/.

## 5 PERSPECTIVES AND CONCLUSION

Data lakes such as [9, 11, 12, 14] 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. Our demonstration showcases ConnectionStudio's novel features for exploring and analyzing ConnectionLens datalakes, following requirements expressed by journalists; using ConnectionStudio, they can discover data structure at a glance, extract customized subsets of a dataset, build up potentially complex queries across several datasets with the help of drop-down menus, and iteratively improve the graph data quality through as-you-go cleaning. Our follow-up work includes devising interactive data graph exploration techniques, moving between the node and entity granularity, as well as relevance ranking of paths and trees between named entities.

## REFERENCES


[1] S Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*. Addison-Wesley, 1995.

[2] Angelos Anadiotis, Oana Balalau, Théo Bouganim, et al. Empowering investigative journalism with graph-based heterogeneous data management. *IEEE DEBull.*, 2021.

[3] Angelos Anadiotis, Oana Balalau, Catarina Conceicao, et al. Graph integration of structured, semistructured and unstructured data for data journalism. *Inf. Systems*, 104, 2022.

[4] Angelos Christos Anadiotis, Ioana Manolescu, and Madhulika Mohanty. Integrating Connection Search in Graph Queries. In *ICDE*, 2023.

[5] Nelly Barret, Simon Ebel, Théo Galizzi, Ioana Manolescu, and Madhulika Mohanty. User-friendly exploration of highly heterogeneous data lakes. In *CoopIS*, 2023.

[6] Nelly Barret, Antoine Gauquier, Jia-Jean Law, and Ioana Manolescu. Exploring heterogeneous data graphs through their entity paths. In *ADBIS*, 2023.

[7] Nelly Barret, Ioana Manolescu, and Prajna Upadhyay. Computing generic abstractions from application datasets. In *EDBT*, 2024.

[8] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J. Miller. Semantics-aware dataset discovery from data lakes with contextualized column-based representation learning. *PVLDB*, 2023.

[9] Corinna Giebler, Christoph Gröger, Eva Hoos, Holger Schwarz, and Bernhard Mitschang. Leveraging the data lake: Current state and challenges. In *DaWaK*, 2019.

[10] Corinna Giebler, Christoph Gröger, Eva Hoos, Holger Schwarz, and Bernhard Mitschang. Modeling data lakes with data vault: Practical experiences, assessment, and lessons learned. In *ER*, 2019.

[11] Rihan Hai, Sandra Geisler, and Christoph Quix. Constance: An intelligent data lake system. In *SIGMOD*, 2016.

[12] Rihan Hai, Christos Koutras, Christoph Quix, and Matthias Jarke. Data lakes: A survey of functions and systems. *TKDE*, 2023.

[13] Maximilian Kuschewski, David Sauerwein, Adnan Alhomssi, and Viktor Leis. BtrBlocks: Efficient columnar compression for data lakes. *SIGMOD*, 2023.

[14] Fatemeh Nargesian, Erkang Zhu, Renée J Miller, Ken Q Pu, and Patricia C Arocena. Data lake management: challenges and opportunities. *PVLDB*, 2019.