Exploiting Formal Concept Analysis for Data Modeling in Data Lakes

Anes Bendimerad, Romain Mathonat, Youcef Remil, Mehdi Kaytoue

Exploiting Formal Concept Analysis for Data Modeling in Data Lakes

Anes Bendimerad[1], Romain Mathonat[1], Youcef Remil[1], and Mehdi Kaytoue[1]

Infologic R&D, 26500 Bourg-Lès-Valence, France
mka@infologic.fr

Abstract. Data lakes are widely used to store extensive and heterogeneous datasets for advanced analytics. However, the unstructured nature of data in these repositories introduces complexities in exploiting them and extracting meaningful insights. This motivates the need of exploring efficient approaches for consolidating data lakes and deriving a common and unified schema. This paper introduces a practical data visualization and analysis approach rooted in Formal Concept Analysis (FCA) to systematically clean, organize, and design data structures within a data lake. We explore diverse data structures stored in our data lake at Infologic, including InfluxDB measurements and Elasticsearch indexes, aiming to derive conventions for a more accessible data model. Leveraging FCA, we represent data structures as objects, analyze the concept lattice, and present two strategies—top-down and bottom-up—to unify these structures and establish a common schema. Our methodology yields significant results, enabling the identification of common concepts in the data structures, such as "resources" along with their underlying shared fields (*timestamp*, *type*, *usedRatio*, etc.). Moreover, the number of distinct data structure field names is reduced by 54% (from 190 to 88) in the studied subset of our data lake. We achieve a complete coverage of 80% of data structures with only 34 distinct field names, a significant improvement from the initial 121 field names that were needed to reach such coverage. The paper provides insights into the Infologic ecosystem, problem formulation, exploration strategies, and presents both qualitative and quantitative results. The source code and datasets of this work are made available: https://zenodo.org/records/10589722

Keywords: Formal Concepts Analysis · Data Lakes · Data Engineering.

1 Introduction

Organizations increasingly rely on data lakes [25] as versatile repositories to store vast and heterogeneous datasets for advanced analytics. The flexibility and scalability offered by data lakes have positioned them as a bedrock for managing massive volumes of raw, unstructured, and heterogeneous data. As defined in [17], a data lake is a massive collection of datasets that: (1) may be hosted in different storage systems; (2) may vary in their formats; (3) may not be accompanied by any useful metadata or may use different formats to describe their metadata; and

(4) may change autonomously over time. At Infologic [5], our data lake serves as a key component in the scope of our predictive maintenance system [9], enabling seamless aggregation of continuously collected data from diverse sources at a low cost. New data collections can be easily added to the data lake by different teams, without the need of defining a priori the data schema. Nevertheless, the unstructured and heterogeneous nature of the data stored in data lakes poses a significant challenge, hindering the full exploitation of their inherent value [17]. It becomes difficult to have a clear understanding of the content of the data and to implement data pipelines that generalize well. Furthermore, executing some common analytics operations between data structures, such as merges and joins, becomes cumbersome, as these structures may use different names for the same fields. Particularly at Infologic, we were using two storage systems, InfluxDB [4] and Elasticsearch [3], each adhering to distinct conventions. This combination of two storage systems were motivated by the effectiveness of InfluxDB in handling metrics and time series, while Elasticsearch is more efficient as a search engine, especially in textual data and JSON documents.

We aim to explore the data structures within our data lake, and extract a set of conventions to consolidate our data model. The data structures under examination include the schemas of InfluxDB measurements and Elasticsearch indexes. Deriving a common schema is a problem that has interested many practitioners. Notably, the Elasticsearch community has proposed the ECS [2] (Elastic Common Schema) to define a common set of fields to be used when storing events data in Elasticsearch. The Common Event Format [7] (CEF) has been designed to propose standard naming conventions for logs in network and security devices and computer systems. Inspired by these well-established standards, we seek to derive a tailored data model that not only aligns with industry practices, but also accommodates the specificities of our business at Infologic. Such specificities concern the architecture that governs our Copilote ERP software, as well as conventions that are already followed in the Relational database that is used to store Copilote critical business data.

This paper addresses this challenge through a comprehensive exploration of a novel approach grounded in Formal Concept Analysis (FCA) [12, 27], aimed at systematically cleaning, structuring, and designing the data within our data lake. We perform interactive data analysis, leveraging the concept lattice as a central tool. FCA has been exploited to address various challenges in both software engineering and data engineering, such as mining functional dependencies for SQL data refractoring [8,15], creating and merging of ontology top-levels [13], and fault localization in software [10]. However, our paper represents the first attempt to employ the concept lattice as a visual tool for consolidating structures in data lakes. We represent data structures, including tables, measurements, and indexes, as objects within a formal context. Each of these objects is described by Boolean attributes that indicate whether a field is present in the related data structure. Subsequently, we derive and analyze the concept lattice from this formal context. In our exploration of this lattice, we present two distinct strategies—top-down and bottom-up—that leverage visual insights to

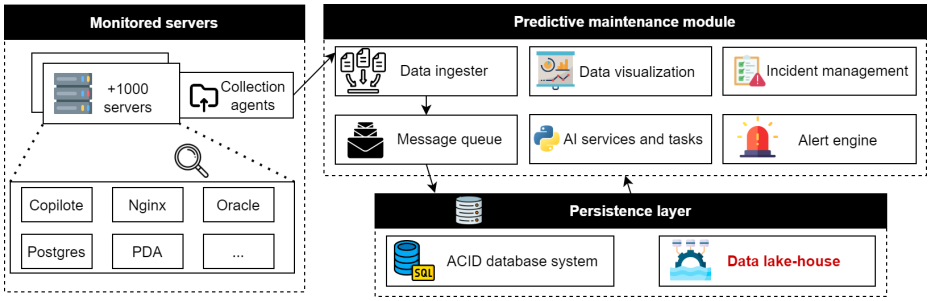


Fig. 1: Simplified architecture of predictive maintenance at Infologic [9].

unify field names and establish a common schema. Our methodology yields significant results, enabling the identification and unification of common concepts in the data structures, such as "resources" along with their underlying shared fields (*timestamp*, *type*, *usedRatio*, etc.). Moreover, the number of data structure field names is reduced by 54%, from 190 to 88 in a subset of our data lake. We achieve a complete coverage of 80% of data structures in this subset with only 34 distinct field names, a significant improvement from the initial 121 field names that were needed to reach such coverage.

Outline. Section 2 provides an overview of Infologic and its ERP software, Copilote, accompanied by a description of our data lake that is used to store predictive maintenance data. Section 3 formulates the studied dataset and problem within the FCA framework, and describes the generation of the concept lattice by illustrating the process with a toy example. In Section 4, we show the used strategies to explore the concept lattice and derive insights that guide us in building our data model. Section 5 approaches the final results from a qualitative and quantitative points of view. Section 6 provides a conclusion and future avenues.

2 Background

2.1 Infologic

Infologic [5] is a leading provider of enterprise resource planning (ERP) solutions for the agri-food, health nutrition, and cosmetic sectors in France. Its flagship product, Copilote, is an ERP software designed to optimize and automate a large panel of business processes, including sales tracking, supply chain and customer relationship management. Infologic provides maintenance of the ERP instances and infrastructure in operation for its clients. As the proper functioning of their businesses depends heavily on the reliable performance and accessibility of the ERP, it is crucial to ensure high availability and excellent maintenance for Copilote. To this aim, Infologic has made substantial investments in a predictive maintenance project [9, 19–23]. In [9], the architecture of this project

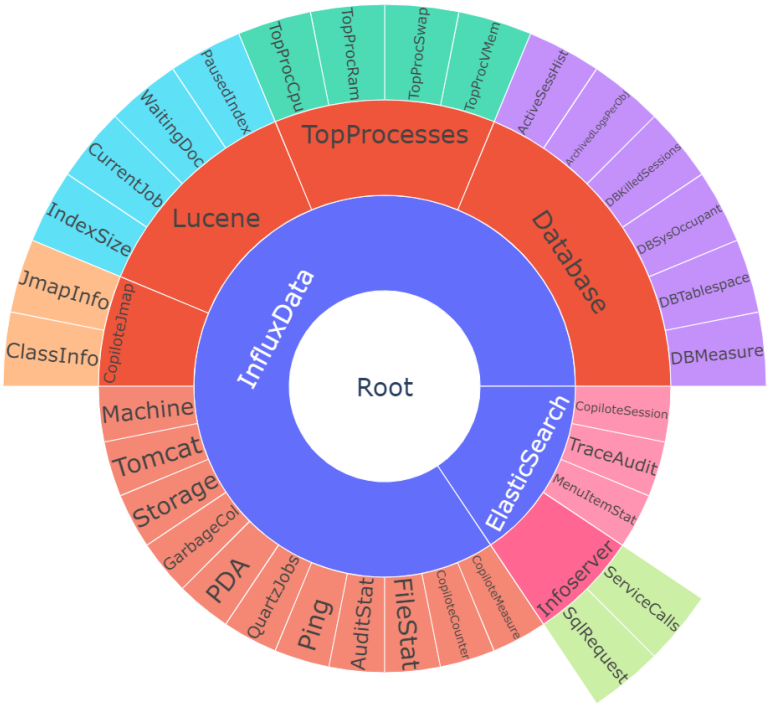


Fig. 2: Data structures stored in our data lake and studied in this paper.

has been presented, and its key components have been described in details. Figure 1 provides a simplified overview of this architecture. One of the foundational components of this project is the data lake, or its broader iteration, the data lake-house [14, 28], incorporating data warehousing features [6, 26].

2.2 Data lake

During the period of our study, the primary components of our data lake comprised InfluxDB [4] and Elasticsearch [3], serving as repositories for the continuous collection and storage of diverse datasets. In Figure 2, we present a subset of data structures contained in our data lake, which constitutes the focus of our analysis. This figure depicts our data as a hierarchy whose leaves represent specific data structures, such as *Machine*, *Tomcat*, *Storage*, among others. The internal nodes in the figure denote cohesive groups of structures belonging to distinct domains. For instance, *Lucene* forms a group encompassing four InfluxDB tables (measurements) designed for storing Lucene monitoring data [16], including $PausedIndex$, $WaitingDoc$, $CurrentJob$, and $IndexSize$. Time series data were stored in InfluxDB, whereas Elasticsearch was dedicated to textual data and JSON documents. The examined subset comprises 32 data structures

Table 1: Toy example of data structures with their underlying fields.

$\mathcal{G}$ (structures)	time	timestamp	used	max	path	name	serviceName	duration
Storage	×		×	×	×			
DBTablespace	×		×	×		×		
ServiceCall		×				×	×	×

(InfluxDB measurements and Elasticsearch indexes) utilizing 190 distinct field names. Some of these names are used in several data structures.

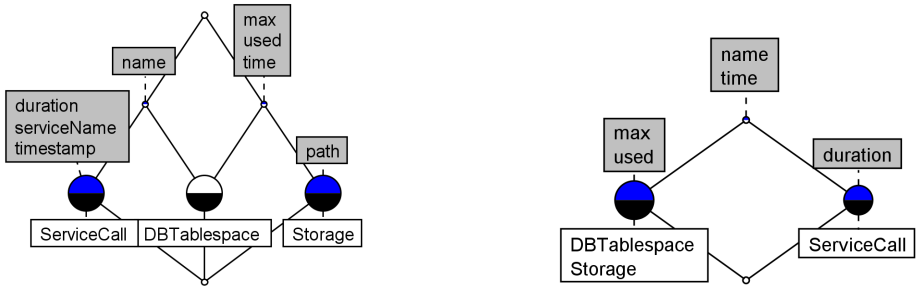
3 Problem Formulation

3.1 Data model with FCA

We formalize the dataset describing our data structures based on Formal Concept Analysis (FCA) [12, 27]. We define the formal context [12] as a triple $\mathbb{K} = (\mathcal{G}, \mathcal{M}, \mathcal{I})$ comprising two sets $\mathcal{G}$ and $\mathcal{M}$ and an incidence relation $\mathcal{I}$ between $\mathcal{G}$ and $\mathcal{M}$. Elements of $\mathcal{G}$ are called objects, and elements of $\mathcal{M}$ are called attributes. To signify that an object $g \in \mathcal{G}$ has an attribute $m \in \mathcal{M}$, we use the notation $g\mathcal{I}m$. Table 1 reports a formal context $(\mathcal{G}, \mathcal{M}, \mathcal{I})$ where objects in $\mathcal{G}$ represent data structures from the data lake, and attributes in $\mathcal{M}$ denote fields within these data structures. The incidence relation $\mathcal{I}$ is visually depicted by crosses in the table, and it represents the fact that a data structure contains a field. For instance, we have "*Storage* $\mathcal{I}$ *used*" that can be read as: "the *Storage* data structure contains the field *used*". In total, the data structure *Storage* is characterized by the following fields: $time, used, max, path$. Notably, some fields in data structures of Table 1 convey similar meanings but are designated by distinct names, such as *time* and *timestamp*, or $serviceName$ and *name*. The goal of our study is to analyze a comprehensive set of data structures, and identify groups of akin field names that manifest recurrently and signify the same underlying notion. Subsequently, we aim to generalize these field names uniformly, establishing a cohesive and unified schema.

Two fundamental operators, namely extent and intent, are defined on a formal context $\mathbb{K} = (\mathcal{G}, \mathcal{M}, \mathcal{I})$. The extent operator, denoted ext, associates to each subset of attributes $B \subseteq \mathcal{M}$ the set of objects $g \in \mathcal{G}$ possessing all attributes in B, that is, $ext(B) = \{g \in \mathcal{G} \mid (\forall m \in B)\, g\,\mathcal{I}\,m\}$. Dually, the intent operator, denoted int, associates to each subset of objects $A \subseteq \mathcal{G}$ the set of attributes $m \in \mathcal{M}$ shared among the objects in A, that is, $int(A) = \{m \in \mathcal{M} \mid (\forall g \in A)\, g\,\mathcal{I}\,m\}$. It is noteworthy that, for $B \subseteq \mathcal{M}$ and $A \subseteq \mathcal{G}$, the following relationships hold: $ext(B) = \bigcap_{m \in B} ext(\{m\})$ and $int(A) = \bigcap_{g \in A} int(\{g\})$. A key theorem in FCA (Proposition 10 in [12]) is:

Theorem 1. *The pair of functions* (ext, int) *form a Galois connection between the power set lattices* $(2^{\mathcal{G}}, \subseteq)$ *and* $(2^{\mathcal{M}}, \subseteq)$*. That is,* $ext \circ int$ *and* $int \circ ext$ *are closure operators on* $(2^{\mathcal{G}}, \subseteq)$ *and,* $(2^{\mathcal{M}}, \subseteq)$ *respectively.*



(a) Before consolidating field names.

(b) After consolidating field names.

Fig. 3: The concept lattice before and after unifying fields from Table 1.

Following this theorem, we can build a concept lattice $(\mathfrak{B}(\mathbb{K}), \leq)$. Elements of $\mathfrak{B}(\mathbb{K})$ are formal concepts and are of the form $(A, B) \in 2^{\mathcal{G}} \times 2^{\mathcal{M}}$ with $A = ext(B)$ and $B = int(A)$. In Table 1, $ext(\{time, used, max\}) = \{Storage, DBTablespace\}$, meaning that objects possessing the fields $time$, $used$, and max are $Storage$ and $DBTablespace$. Dually, $int(\{Storage, DBTablespace\}) = \{time, used, max\}$, indicating that the common fields between $Storage$ and $DBTablespace$ are $time$, $used$, and max. Since $int(ext(\{time, used, max\})) = \{time, used, max\}$, the pair $(\{Storage, DBTablespace\}, \{time, used, max\})$ is a formal concept.

3.2 Concept lattice

We construct the concept lattice $(\mathfrak{B}(\mathbb{K}), \leq)$ from our formal context $\mathbb{K}$. Various tools can be used to visualize the concept lattice given any formal context stored in some specific format [24], such as a CSV file. In our study, Concept Explorer [1] was utilized for this purpose. Figure 3 (a) shows the concept lattice generated from the toy dataset in Table 1. Each node of this lattice represents a formal concept $(A, B) \in 2^{\mathcal{G}} \times 2^{\mathcal{M}}$ such that $A = ext(B)$ and $B = int(A)$. For example, the right child of the root corresponds to the formal concept $(\{Storage, DBTablespace\}, \{time, used, max\})$. Subsequently, the right child of the latter formal concept is $(\{Storage\}, \{time, used, max, path\})$, which is a formal concept covering only the object (data structure) $Storage$. Using such data visualization, our aim is to analyze concepts in order to derive relevant unification of fields and structures. In the toy dataset, we can unify the field names $time$ and $timestamp$, renaming both as $time$. We can also consolidate the field names $serviceName$, $name$, and $path$ into a unified label, such as $name$. These transformations result in the unified field names ascending in the direction of the top of the lattice. Figure 3 (b) illustrates the final lattice after applying these transformations on the toy formal context of Table 1. The two fields $name$ and $time$ have ascended to the root of the lattice since they are covered by all the objects $Storage$, $DBTablespace$, and $ServiceCall$. An alternative method

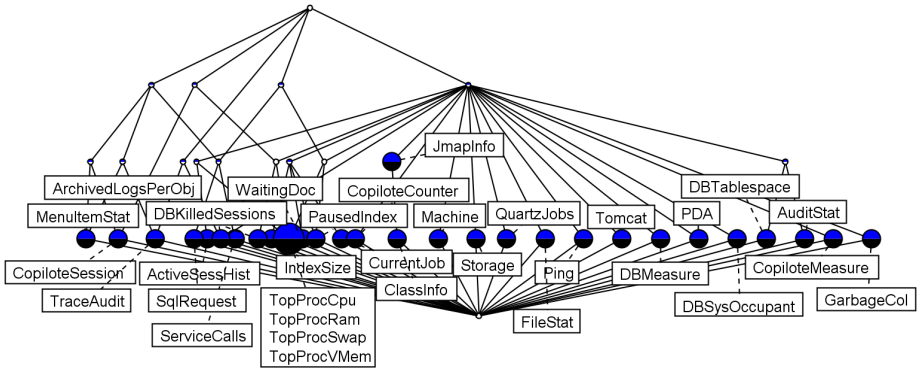


Fig. 4: Concept lattice depicting data structures within our data lake. The objects (data structures) are indicated in the lattice.

to consolidate the data model is to exploit association rules derived from the formal context. However, we believe that the visual exploitation of the lattice is easier for FCA non-practitioners. We show in Section 4 that extracting relevant insights is simplified when viewing the concept lattice.

4 Exploiting the Concept Lattice

4.1 Analyzing the initial lattice

We generate the concept lattice from the dataset depicted in Figure 2, comprising 32 objects (data structures) and 191 attributes (field names). Figure 4 presents the complete lattice, highlighting the objects associated with each formal concept, while Figure 5 displays the attributes within each concept. The root represents the formal concept that covers all the objects of the dataset. Since there is no field name that is present in all the data structures, the root node in Figure 5 has an empty intent. Below the root node, we observe a few nodes that sit in this second level of the lattice, such as the concept with the intent $\{instanceType, instanceCode, \_time\}$. This concept reflects a set of field names that are present in every data structure of InfluxDB, but they are not used in Elasticsearch. Other examples of concepts as children of the root include those with the intent $\{rank\}$ and $\{user\}$. In Figure 4, we notice that nearly all objects (data structures) reside in the penultimate level. Notably, this lattice exhibits a small height and a large width due to the limited number of common field names between data structures, attributed to the absence of a standardized naming convention. Consequently, there is a limited number of internal formal concepts representing shared attributes. This lattice serves as the basis for our analysis, where we explore two distinct data analysis approaches outlined in this section: the top-down approach and the bottom-up approach.

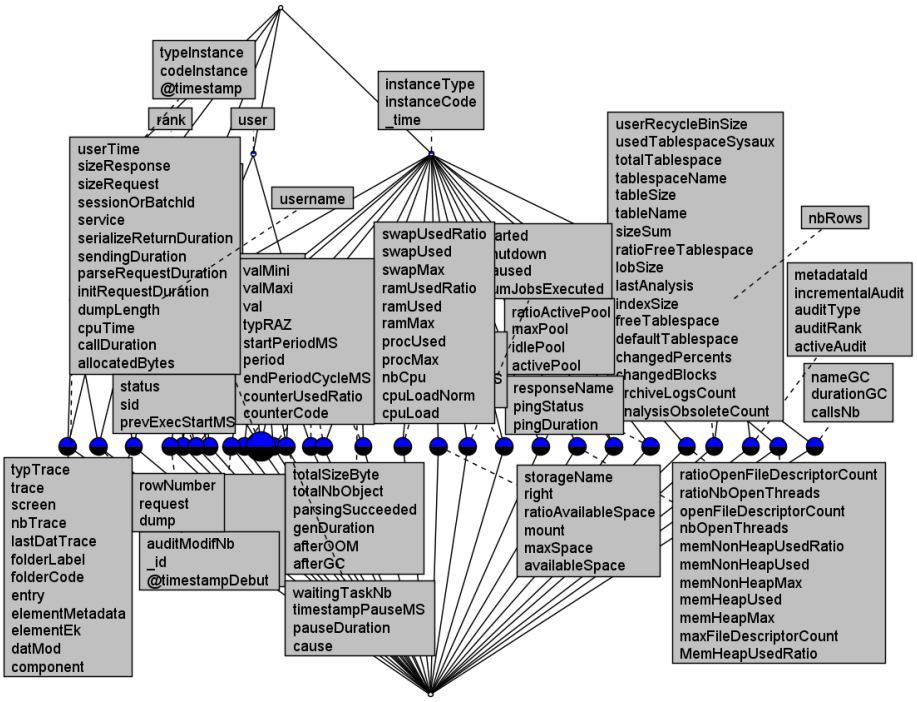


Fig. 5: Concept lattice depicting data structures within our data lake. The attributes (field names) are indicated in the lattice.

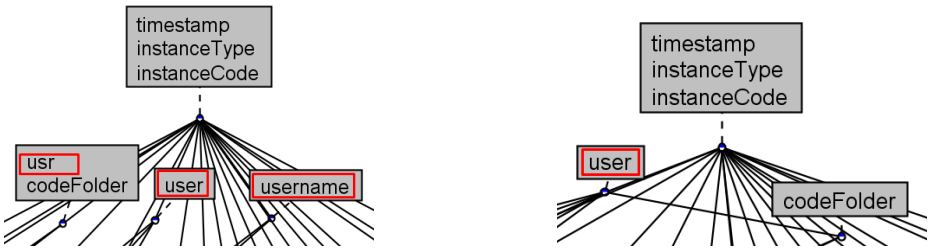
4.2 The top-down approach

This approach consists in starting from the root (the top-node) of the hierarchy and find akin field names that can be unified. In this section, we illustrate this top-down approach with two interesting findings that were made possible thanks to this data exploration.

Unifying generic fields from InfluxDB and Elasticsearch. As depicted in Figure 6 (a), the concept lattice visualization provides a clear view of the root having two children covered by similar sets of fields. The first group, comprising $\{typeInstance, codeInstance, @timestamp\}$, is uniformly present in all data structures stored in Elasticsearch. Here, $typeInstance$ signifies the Copilote ERP instance type (production, test, development, deployment), $instanceCode$ is a unique identifier for Copilote instances, and $@timestamp$ denotes the creation time of the collected event or metric. Simultaneously, the second group $\{instanceType, instanceCode, \_time\}$ is utilized across all InfluxDB measurements. These two groups have been consolidated into the unified field names $\{timestamp, instanceType, instanceCode\}$. As shown in Figure 6 (b), this group

(a) Before unifying generic fields. (b) After unifying generic fields.

Fig. 6: Top-down approach. Unifying generic fields from InfluxDB and Elasticsearch.



(a) Before unifying user fields. (b) After unifying user fields.

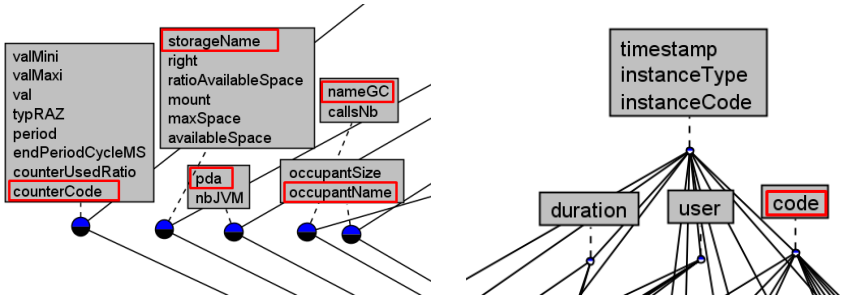
Fig. 7: Top-down approach. Unifying *user* fields.

has ascended to the root of the lattice, since it is covered by all our data structures (all the objects of the formal context).

Unifying the *user* field. Another compelling application of the top-down approach involves the consolidation of the field describing the user identifier, as illustrated in Figure 7 (a). Initially, we observed distinct field names referencing the user identifier, namely *usr*, *user*, and *username*. Each of them is covered by several data structures, which makes them visible in the upper layers of the lattice. We have unified these identifiers under a standardized field name, *user*, as depicted in Figure 7 (b).

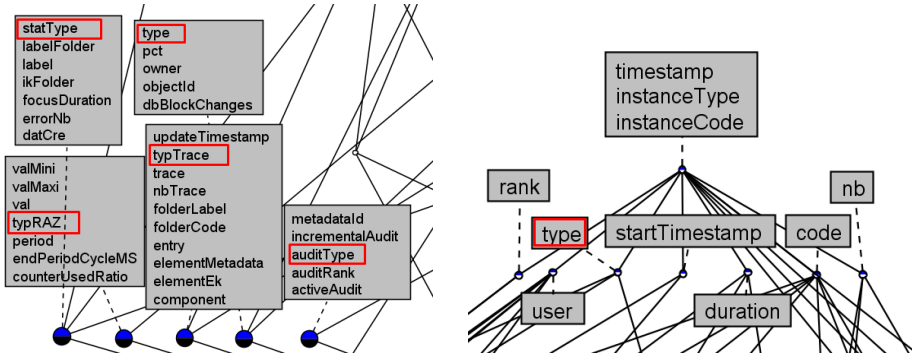
4.3 The bottom-up approach

Another method to exploit concept lattice visualization is to inspect the bottom layers to discern a substantial set of distinct fields that are used in a relatively limited set of data structures but represent the same notion. Unlike the top-down approach, which unifies groups of fields already used in many data structures, the bottom-up approach has another focus. It targets the unification of fields generally employed in unique data structures and, consequently, found only in



(a) Before unifying code fields. (b) After unifying code fields.

Fig. 8: Bottom-up approach. Unifying *code* fields.

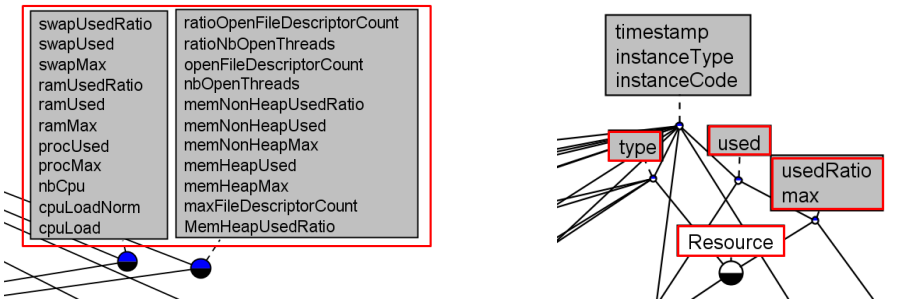


(a) Before unifying type fields. (b) After unifying type fields.

Fig. 9: Bottom-up approach. Unifying *type* fields.

the last layers of the lattice. We illustrate with three noteworthy results that have been discovered through this methodology.

Unifying the *code* attribute. By examining data structures in the last layer, we observe that many of them have a field dedicated to storing the name of a component monitored by the respective data structure. Figure 8 (a) illustrates this observation with five data structures. For instance, *storageName* denotes the name of the storage space monitored by metrics such as *maxSpace* and *availableSpace*. Similarly, *nameGC* represents the name of the Java Garbage Collector under surveillance, for which the number of executions (*callsNb*) is logged. We have unified these fields under the same name, *code*, which subsequently ascended to a position beneath the root of the lattice, as shown in Figure 8 (b).



(a) Before unifying resources fields. (b) After unifying resources fields.

Fig. 10: Bottom-up approach. Unifying resources fields in a *Resource* table.

Unifying the *type* attribute. An information that is present in a wide range of data structures is the type of the component described by the data. Figure 9 (a) showcases five selected examples of such fields: $statType$, $type$, $typRAZ$, $typTrace$, $auditType$. As reported in Figure 9 (b), all of these fields have been consolidated under the same name: $type$.

Unifying the *Resource* table. In Figure 10, we report a specific optimization of data model. We show two data structures that describe resource utilization of the machine hosting the ERP, as well as its JVM. For example, $swapUsed$ indicates the used swap in bytes, $swapMax$ indicates the limit of the swap size, while $swapUsedRatio$ indicates the proportion of the used swap. Similar fields exist to describe other resources, such as RAM, Java heap space, Java non heap space, and more. We have consolidated these fields into a new data structure named *Resources*. Within this structure, the *type* field indicates the resource being monitored (e.g., "swap", "ram", "heap"). Subsequently, the utilization of the resource indicated in the *type* field is described by three generic fields: *used*, *max*, and *usedRatio*. Figure 10 (b) reveals that all of these fields have ascended to the upper layers of the lattice, consequently elevating the entire *Resources* structure within the lattice.

5 Results

In this section, we begin by reporting and analyzing the final lattice derived from the dataset after performing our data structure consolidation. We highlight the distinctions between this final lattice and the one generated from the original dataset. Then, we perform a quantitative study to analyze the distribution of field names, their coverage of the data structures, and measure the improvement achieved from this perspective.

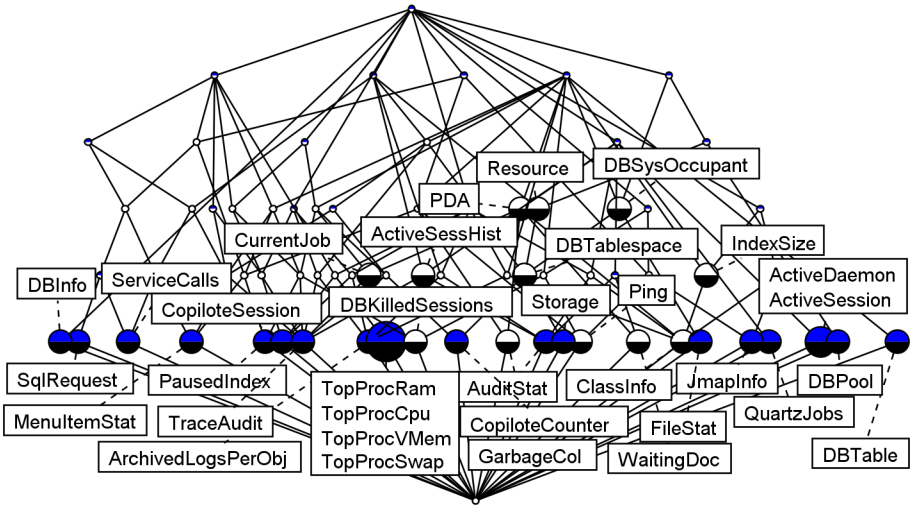


Fig. 11: The concept lattice of our data structures after consolidating field names. The objects (data structures) are indicated in the lattice.

5.1 Final lattice

Figure 11 illustrates the concept lattice generated from the final dataset after applying all the transformations related to the proposed data model. The illustration also includes the objects associated to each of the formal concepts represented in the lattice. Some data structures have moved upward in the lattice, as they use only common fields without any specific field names. For example, *Resource* emerges as one of the most generic data structures, positioned at one of the initial layers of the lattice. Figure 12 presents the same lattice but with a view on field names (attributes of the formal context). Three field names appear in the root of the lattice—*timestamp*, *instanceType*, and *instanceCode*—as they are covered by all the data structures. In the second layer, we find generic fields such as *user*, *type*, *duration*, *code*, *used*. Notably, the concept covered by *used* represents resources whose consumption is measurable, and the field *used* is employed to store the quantity of utilization of the related resource. For example, if the resource is the swap memory, *used* indicates the number of bytes consumed by the swap. Furthermore, the concept covered by *used* has a child that is extended with the fields *max* and *usedRatio*, to represent resources whose capacity is limited and known. Another interesting observation in the lattice is that, while *duration* sits in the second layer, the fields *startTimestamp* and *endTimestamp* are separated in other concepts that come lower in the lattice. A possible further consolidation of our model is to add the fields *startTimestamp* and *endTimestamp* to all the "events" data structures that are described by a *duration*. This would unify the three fields *duration*, *startTimestamp* and *endTimestamp* in a same formal concept. Following our data structure consoli-

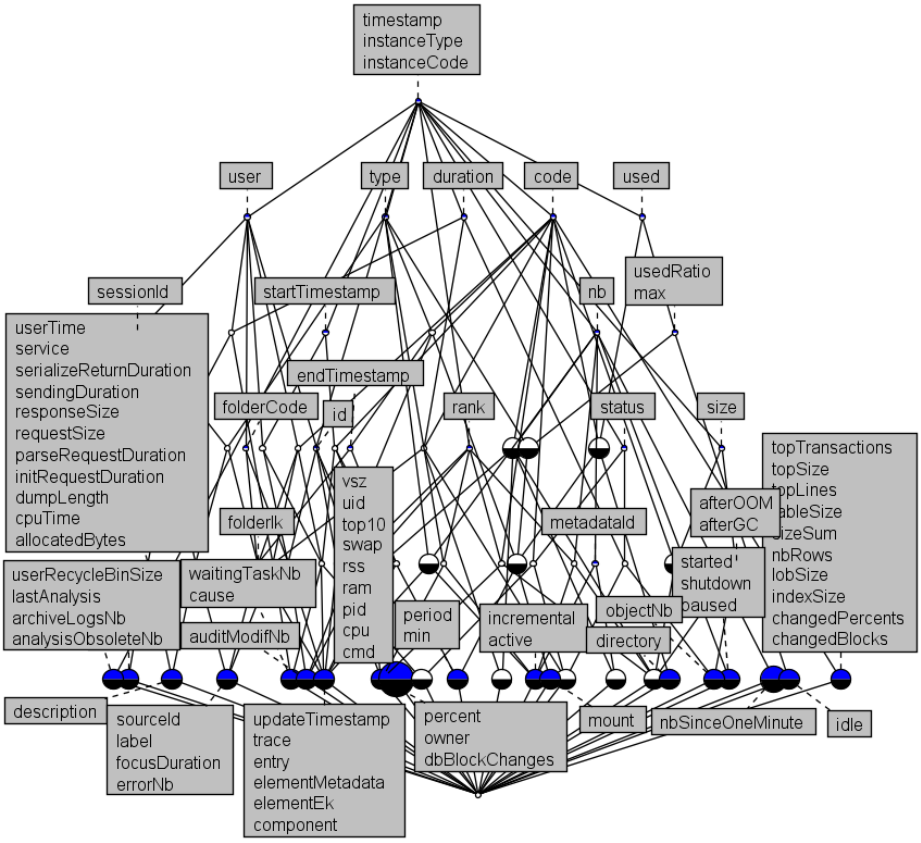


Fig. 12: The concept lattice of our data structures after consolidating field names. The attributes (field names) are indicated in the lattice.

dation, the number of distinct field names has decreased from 190 to 88. However, the number of formal concepts has increased from 44 to 72, due to more sets of fields common between data structures. Consequently, the lattice height has increased from initially 4 to 6 after performing the data structure consolidation.

5.2 Analyzing the number of attributes

Our objective is to assess the extent of data structures coverage by the new fields names, and compare it with their coverage before applying our transformations. To achieve this, we sort the field names in descending order based on the frequency of their utilization in data structures. Subsequently, we measure the proportion of completely covered data structures by a given number of top fields ranked with respect to their utilization frequency. Figure 13 reports the results. In the final context, we achieve a coverage of 75% of data structures with 25 field names, marking a significant improvement compared to the initial

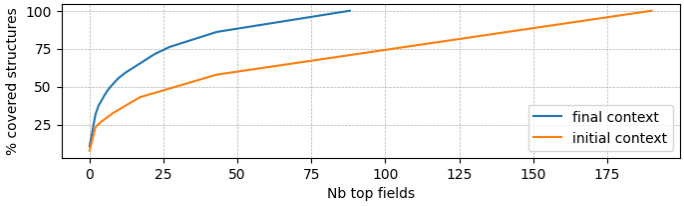


Fig. 13: Covered data structures given the number of top field names used.

dataset where the same number of field names covered less than 50% of the data structures. Moreover, we can cover the entire dataset with 88 field names instead of original 190 field names, reducing them by 54%.

6 Conclusion

In conclusion, our application of Formal Concept Analysis (FCA) to the exploration of data lake structures at Infologic has proven highly effective. By systematically analyzing diverse data structures and leveraging FCA's concept lattice, we successfully reduced the number of distinct attributes by 54%, from 190 to 88, and covered 80% of data structures with only 34 distinct field names. This approach not only addresses specific challenges at Infologic, but also provides a valuable framework for organizations navigating the complexities of data lakes. We believe that concept lattices are effective visual tools that are accessible even to individuals who are not data analysis experts. They can read and understand the presented concepts and make informed decisions accordingly. Moving forward, the insights gained pave the way for a cleaner and a more exploitable data lake. Moreover, the resulting unified schema can serve as a model for our Electronic Data Interchange (EDI [18]) module, a generic Copilote component serving as an interface for the exchange of data between Copilote instances and other systems. As a part of future work, a promising avenue is to incorporate advanced tools to enhance and automate our manual data analysis methods used in this paper. Integrating Natural Language Processing (NLP) and graph mining techniques will empower us to identify groups of similar fields, enabling the mapping of disparate fields to a unified name. Additionally, leveraging analogy-based reasoning will assist in identifying corresponding fields across different tables. For example, the following analogy would make it possible to unify the fields a and b: "the field a is to the table A as the field b is to the table B". Thanks to these techniques, our approach would be able to scale to larger data lakes with a higher number of data structures to consolidate. Another possibility to improve the scalability of our approach is to exploit AOC-posets [11], which are smaller and more concise subsets of concept lattices.

Acknowledgement

This research was supported by the ANR project "Formal Concept Analysis: A Smart Tool for Analyzing Complex Data" (SmartFCA), ANR-21-CE23-0023. The authors also express their sincere gratitude to Philippe Cancellier and Guillaume Kheng for their contributions to the datalake at Infologic and to Amedeo Napoli for insightful discussions.

References

1. The concept explorer (2024), https://conexp.sourceforge.net/
2. Elastic common schema (2024), https://www.elastic.co/guide/en/ecs/current/ecs-reference.html
3. Elasticsearch (2024), https://www.elastic.co/
4. Influxdb (2024), https://www.influxdata.com/
5. Infologic-copilote (2024), https://www.infologic-copilote.fr/
6. Anisimov, A.A.: Review of the data warehouse toolkit: the complete guide to dimensional modeling (2nd edition) by ralph kimball, margy ross. john wiley & sons, inc. 2002. SIGMOD Rec. **32**(3), 101–102 (2003). https://doi.org/10.1145/945721.945741
7. ArcSight, H.: Common event format. Tech. rep., tech. rep., July (2009)
8. Baixeries, J., Kaytoue, M., Napoli, A.: Characterizing functional dependencies in formal concept analysis with pattern structures. Ann. Math. Artif. Intell. **72**(1-2), 129–149 (2014). https://doi.org/10.1007/S10472-014-9400-3
9. Bendimerad, A., Remil, Y., Mathonat, R., Kaytoue, M.: On-premise aiops infrastructure for a software editor SME: an experience report. In: Chandra, S., Blincoe, K., Tonella, P. (eds.) Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023. pp. 1820–1831. ACM (2023). https://doi.org/10.1145/3611643.3613876
10. Cellier, P., Ducassé, M., Ferré, S., Ridoux, O.: Formal concept analysis enhances fault localization in software. In: Medina, R., Obiedkov, S.A. (eds.) Formal Concept Analysis, 6th International Conference, ICFCA 2008, Montreal, Canada, February 25-28, 2008, Proceedings. Lecture Notes in Computer Science, vol. 4933, pp. 273–288. Springer (2008). https://doi.org/10.1007/978-3-540-78137-0_20
11. Dolques, X., Ber, F.L., Huchard, M.: Aoc-posets: a scalable alternative to concept lattices for relational concept analysis. In: Ojeda-Aciego, M., Outrata, J. (eds.) Proceedings of the Tenth International Conference on Concept Lattices and Their Applications, La Rochelle, France, October 15-18, 2013. CEUR Workshop Proceedings, vol. 1062, pp. 129–140. CEUR-WS.org (2013), https://ceur-ws.org/Vol-1062/paper11.pdf
12. Ganter, B., Wille, R.: Formal Concept Analysis. Springer (1999)
13. Ganter, B., Stumme, G.: Creation and merging of ontology top-levels. In: de Moor, A., Lex, W., Ganter, B. (eds.) Conceptual Structures for Knowledge Creation and Communication, 11th International Conference on Conceptual Structures, ICCS 2003 Dresden, Germany, July 21-25, 2003 Proceedings. Lecture Notes in Computer Science, vol. 2746, pp. 131–145. Springer (2003). https://doi.org/10.1007/978-3-540-45091-7_9, https://doi.org/10.1007/978-3-540-45091-7_9

14. Harby, A.A., Zulkernine, F.: From data warehouse to lakehouse: A comparative review. In: 2022 IEEE International Conference on Big Data (Big Data). pp. 389–395. IEEE (2022). https://doi.org/10.1109/BigData55660.2022.10020719
15. Lopes, S., Petit, J., Lakhal, L.: Functional and approximate dependency mining: database and FCA points of view. J. Exp. Theor. Artif. Intell. **14**(2-3), 93–114 (2002). https://doi.org/10.1080/09528130210164143
16. Lucene, A.: Apache lucene-overview. Internet: http://lucene. apache. org/iava/docs/[Jan. 15, 2009] (2010)
17. Nargesian, F., Zhu, E., Miller, R.J., Pu, K.Q., Arocena, P.C.: Data lake management: Challenges and opportunities. Proc. VLDB Endow. **12**(12), 1986–1989 (2019). https://doi.org/10.14778/3352063.3352116
18. Premkumar, G.P., Ramamurthy, K., Nilakanta, S.: Implementation of electronic data interchange: An innovation diffusion perspective. J. Manag. Inf. Syst. **11**(2), 157–186 (1994). https://doi.org/10.1080/07421222.1994.11518044
19. Remil, Y.: A data mining perspective on explainable AIOps with applications to software maintenance. Ph.D. thesis, INSA de Lyon (2023)
20. Remil, Y., Bendimerad, A., Chambard, M., Mathonat, R., Plantevit, M., Kaytoue, M.: Mining java memory errors using subjective interesting subgroups with hierarchical targets. arXiv preprint arXiv:2310.00781 (2023)
21. Remil, Y., Bendimerad, A., Mathonat, R., Chaleat, P., Kaytoue, M.: "what makes my queries slow?": Subgroup discovery for SQL workload analysis. In: 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021. pp. 642–652. IEEE (2021). https://doi.org/10.1109/ASE51524.2021.9678915
22. Remil, Y., Bendimerad, A., Mathonat, R., Raïssi, C., Kaytoue, M.: Deeplsh: Deep locality-sensitive hash learning for fast and efficient near-duplicate crash report detection. CoRR **abs/2310.06703** (2023). https://doi.org/10.48550/ARXIV.2310.06703
23. Remil, Y., Bendimerad, A., Plantevit, M., Robardet, C., Kaytoue, M.: Interpretable summaries of black box incident triaging with subgroup discovery. In: 8th IEEE International Conference on Data Science and Advanced Analytics, DSAA 2021, Porto, Portugal, October 6-9, 2021. pp. 1–10. IEEE (2021). https://doi.org/10.1109/DSAA53316.2021.9564164
24. Saab, N., Huchard, M., Martin, P.: Evaluating formal concept analysis software for anomaly detection and correction. In: Cordero, P., Krídlo, O. (eds.) Proceedings of the Sixteenth International Conference on Concept Lattices and Their Applications (CLA 2022) Tallinn, Estonia, June 20-22, 2022., Tallinn, Estonia, June 20-22, 2022. CEUR Workshop Proceedings, vol. 3308, pp. 213–218. CEUR-WS.org (2022), `https://ceur-ws.org/Vol-3308/Paper18.pdf`
25. Sawadogo, P.N., Darmont, J.: On data lake architectures and metadata management. J. Intell. Inf. Syst. **56**(1), 97–120 (2021). https://doi.org/10.1007/S10844-020-00608-7
26. Widom, J.: Review - an overview of data warehousing and OLAP technology. ACM SIGMOD Digit. Rev. **1** (1999), `https://dblp.org/db/journals/dr/Widom99c.html`
27. Wille, R.: Restructuring lattice theory: an approach based on hierarchies of concepts. In: Ordered Sets. pp. 445–470 (1982)
28. Zaharia, M., Ghodsi, A., Xin, R., Armbrust, M.: Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics. In: 11th Conference on Innovative Data Systems Research, CIDR 2021, Virtual Event, January 11-15, 2021, Online Proceedings. www.cidrdb.org (2021)