A Formal Framework for Data Lakes Based on Category Theory

Alexis Guyot, Annabelle Gillet, Eric Leclercq, Nadine Cullot

A Formal Framework for Data Lakes Based on Category Theory

Alexis Guyot
alexis.guyot@u-bourgogne.fr
LIB Univ. Bourgogne Franche Comté
Dijon, France

Annabelle Gillet
annabelle.gillet@u-bourgogne.fr
LIB Univ. Bourgogne Franche Comté
Dijon, France

Éric Leclercq
eric.leclercq@u-bourgogne.fr
LIB Univ. Bourgogne Franche Comté
Dijon, France

Nadine Cullot
nadine.cullot@u-bourgogne.fr
LIB Univ. Bourgogne Franche Comté
Dijon, France

ABSTRACT

The management of Big Data requires flexible systems to handle the heterogeneity of data models as well as the complexity of analytical workflows. Traditional systems like data warehouses have reached their limits due to their rigid schema-on-write paradigm, that requires well identified and defined use cases to ingest data. Data lakes, with their schema-on-read paradigm, have been proposed as more flexible systems in which raw data are directly stored in their original format associated with metadata, to be accessed and transformed only when users need to process or analyze them. Thus, it is necessary to define and control the different levels of abstraction and the dependencies among functionalities of a data lake to use it efficiently. In this article, we present a formal framework aiming to define a data lake pattern and to unify the interactions among the functionalities. We use the category theory as theoretical foundations to benefit from its high level of abstraction and its compositionality. By relying on different categories and functors, we ensure the navigation among the functionalities and allow the composition of multiples operations, while keeping track of the entire lineage of data. We also show how our framework can be applied on a simple example of data lake.

CCS CONCEPTS

• **Information systems** → *Decision support systems*; *Business intelligence*; **Data management systems**; **Information storage systems**; • **Software and its engineering** → *Architecture description languages*; • **Computer systems organization** → *Architectures*;

KEYWORDS

Data Lakes, Category Theory, Architecture Pattern

ACM Reference Format:
Alexis Guyot, Annabelle Gillet, Éric Leclercq, and Nadine Cullot. 2022. A Formal Framework for Data Lakes Based on Category Theory. In *International Database Engineered Applications Symposium (IDEAS'22), August 22–24, 2022, Budapest, Hungary.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3548785.3548797

1 INTRODUCTION

Changes in the way of producing and consuming data have led to the emergence of new issues for information systems. Big Data are especially tied to 5 main concerns, usually referred to 5V: Volume, Variety, Velocity, Veracity and Value. Management of such data requires appropriate environments offering enough flexibility to support heterogeneous data with different models, but also multiple data analysis tools and complex pipelines or workflows. Traditional systems like data warehouses have thereby reached their limits due to their schema-on-write paradigm. Indeed, when using Extract-Transform-Load (ETL) processes to ingest data, the time-consuming and difficult data integration and cleaning steps require well known use cases, that hinder the flexibility of such systems.

Data lakes have been proposed as flexible environments for storing and analyzing Big Data [13] with a schema-on-read paradigm. In other words, raw data are usually directly stored in their original format and only processed when needed, but at the cost of complex model transformations that are supported by users. Different kind of metadata are proposed to help users to navigate, select, validate and transform data according to their needs. To a certain extent, it prevents from turning data lakes into data swamps [20], whose usability is reduced because of the difficulty to localize data.

Data lakes have been described multiple times in the literature [9, 41, 44], but we follow Hai et al.'s definition [25]:

> A data lake is a flexible, scalable data storage and management system, which ingests and stores raw data from heterogeneous sources in their original format, and provides query processing and data analytics in an on-the-fly manner.

This definition is fairly complete. It presents the data lake as an integrated system from the user's perspective (for example a data engineer), exposing multiple functionalities. At a technical level, a data lake is in fact an architecture that brings together specialized components.

Despite being defined as systems for both storing and processing data with management and analytical processes, few architectures of data lakes take all these aspects into consideration in a unified manner [44]. This has led to the creation of alternative systems like Delta Lakes [2], Lakehouses [3] and Data Mesh [10], that focus on improving a subset of functionalities of data lakes. As a result, these systems have major drawbacks: they do not necessarily meet the functional requirements and they lack robustness. One of the main causes is that they are built as an assembly of isolated components having different behavioral properties, and thus it is difficult to ensure the validity of the expected properties for the whole system.

IDEAS'22, August 22–24, 2022, Budapest, Hungary
2022. ACM ISBN 978-1-4503-9709-4/22/08…$15.00
https://doi.org/10.1145/3548785.3548797

However, the use of multiple components to support all the functionalities of the data lake is inevitable. Thus, the notion of data lake is in fact an architecture pattern in which the functionalities are well-defined. To avoid the data lake construction issues, some works narrow their system for a specific use case according to different domains [31, 34, 36, 40, 43]. We adopt a more abstract point of view, and aim to define a framework allowing to generalize the data lake pattern and to unify the component interactions.

The greater control offered by robust systems is usually obtained through solid theoretical foundations [5, 29], such as Codd's relational model [8] and algebra for database management systems, which can provide formal frameworks for studying different properties and their preservation or for building optimization processes. Data lakes, as pragmatic solutions originally designed to resolve industrial issues, were not defined at the time with strong theoretical foundations to describe and validate their functionalities, or to control their uses. Nevertheless, they could benefit from such framework.

Category theory [15] is a meta-mathematical formalism introduced in the 1940s by Saunders MacLane and Samuel Eilenberg. It has already been successfully used for building formal frameworks in various domains of computer science like functional programming or software architectures. These domains especially benefit from the high level of abstraction and compositionality of this theory. Data lakes can greatly take advantage of these characteristics, as data and functions need to be represented altogether. Indeed, the schema-on-read paradigm requires enough expressivity to allow all kind of processing, but it also necessitates constraints in order to control data and metadata organization as well as their transformations and analysis. Category theory can provide all of these requirements.

In this article, we propose to use category theory to build a formal framework allowing the interconnections among the different functionalities of a data lake, and unifying the levels of abstraction. It allows to compose the functionalities, and thus to keep track of the lineage of data, in order to give a formal structure to data lakes while coping with their need for flexibility. We also show the usability of the framework by applying it to an example of simple data lake.

The remainder of this paper is structured as follows. First, we describe in section 2 the main functionalities of data lakes according to the literature, we present other formalisms previously proposed for data lakes and we show how the category theory can contribute to the formalization of these systems. In section 3, we introduce the main concepts of the category theory and our formal framework. We then use it to model a small example of data lake in section 3.3. Finally, we draw conclusions of our work and open up perspectives for the future in section 4.

2 STATE OF THE ART AND DISCUSSION

In this section, after describing the main functionalities of data lakes, we argue that various levels of abstraction and dependencies among functionalities justify the need of a formalism allowing both abstraction and composition. In a second part, we study the major works regarding the formalization of data lake components and we show how category theory can be used in this context.

2.1 Main Functionalities of Data Lakes

Reviews of the literature [9, 25, 41, 44] agree on 4 main functionalities for data lakes, namely Data Storage, Data Ingestion, Data Maintenance and Data Exploration.

Data Storage can take several forms in data lakes, from single systems handling data heterogeneity with generic models [16, 39] to polystores built as an assembly of specialized database management systems [4, 24, 27]. As datasets must be properly and accurately annotated with metadata so that the data lake does not become a data swamp [20], the data storage functionality also includes the storage of metadata.

Data Ingestion provides tools for connecting the system to data sources, loading the data in streaming and/or batch manner and retrieving or producing basic metadata [19, 38]. Some data may require to only store aggregated insights to reduce their important volume, as it is the case for data streams of sensor data with high velocity.

Data Maintenance ensures: 1) the usability of the data by organizing the lake [1, 33] and by extracting more advanced metadata [4, 17, 27], for example through profiling or through the discovery of relationships among datasets; 2) the quality of data [16, 26], by guarantying or improving it, for example through the application of integrity constraints; and 3) the ease of use of the system by providing functionalities that make schema-on-read simpler and more efficient such as schema pre-integration [24].

Finally, **Data Exploration** functionality allows the discovery of content in data lakes through unified query interfaces [4, 24] or navigational algorithms based on measures of relatedness [17] or faceted search [27]. During the exploration phase, datasets are retrieved and integrated in an on-the-fly manner. Queries and algorithms can be applied on them in order to obtain results according to the case study.

The description of the functionalities extracted from the literature reveals two major characteristics of data lakes. Firstly, it brings to light **different levels of abstraction** related to: 1) data, including various models, formats and metadata; 2) software architectures, with different strategies and components that can be used to store and process data; and finally 3) functionalities themselves, composed to implement other higher level services. Secondly, the definitions also reveal existing **dependencies among the main functionalities**. Data exploration depends on data maintenance and on data storage, data storage depends on data ingestion and finally data maintenance depends on data storage.

The higher complexity induced by abstractions and dependencies is a strong motivation for building a formal framework able to ensure the robustness of data lakes and to control the interactions among components.

2.2 On Formalization of Data Lakes

Only few proposals have been made to provide such formal framework for data lakes. Most of previous works have focused on the formalization of mostly isolated aspects of data lakes such as metadata models, data storage, analytical queries, etc.

Several models have been proposed for metadata modeling using UML, entity relationship or graph theory. In [45] the authors state that existing metadata models are either tailored for a specific use

case or not generic enough to be used in different contexts. They extend their previous model called MEDAL (MEtadata model for DAta Lakes) to build a more generic one called goldMEDAL, defined on three levels: conceptual, logical and physical. The conceptual level is formalized through set theory and describes data entities and groupings, as well as hierarchy and lineage relationships. The logical level puts together the previous elements through graph theory concepts, especially nodes, edges and hyperedges. The physical level is finally implemented with the metadata framework Apache Atlas. The overall proposal of goldMEDAL is synthesized in a UML class diagram.

In [42], a classification of metadata in two groups is proposed, with a special attention given to metadata related to data governance concerns such as data access, quality and security. Metadata describing various relationships among datasets are classified as inter-metadata and metadata describing datasets themselves are classified as intra-metadata. The conceptual metadata model is represented with a UML class diagram. A data lake architecture based on three zones is also represented but not formalized. This proposal has been later extended by the authors in [55] with a new analysis-oriented metadata model, also conceptualized through a UML class diagram.

Finally for metadata models, ensemble modeling and more precisely data vaults are used in [37] to create a model allowing better evolutivity for data and schema. At the conceptual level, datasets are classified inside satellites, logically abstracted inside hubs and finally associated inside links. The proposal is represented with a graph.

The storage layer of semantic data lakes has been formalized with set theory in [11] as a tuple containing a set of data sources (datasets), a set of metadata catalogs describing the datasets with directed graphs, a global knowledge graph and a mapping function relating metadata to knowledge concepts in the global graph. In the same article, the authors also propose a set-theoretical formalization of analytical queries. They are defined as sets of indicators of interest measured along sets of dimensions of analysis. A response to such query is a set of metadata and transformation rules allowing the discovery of relevant data.

In [25], the authors compare four formal schema mapping languages based on tuple-generating dependencies (tgds), namely simple tgds, nested tgds, second-order tgds (SO tgds) and plain SO tgds, as potential formal frameworks for integration tasks in data lakes. The different tgds are compared depending on their expressiveness as well as on the set of structural or reasoning properties they can ensure among the existence of universal solutions, closure under target homomorphism and allowing conjunctive query rewriting. SO tgds languages are identified as more expressive than the other two but also less reliable on the properties due to their higher time complexity for model checking.

Despite existing attempts to formalize parts of data lakes, a formal framework allowing a unified and complete representation of all the functionalities and levels of abstraction of such systems is still missing. Moreover, existing pieces of formal definitions are mainly based on semi-formal and descriptive models like UML or labeled graphs, which are not restrictive enough to guarantee mathematical, structural and/or reasoning properties to the proposed model and to ensure their preservation in any subsequent concrete implementation [6].

2.3 Contributions of Category Theory to Data Lakes

Category theory is an abstract meta-mathematical theory. It helps to reconcile the expressiveness of descriptive models and the restrictiveness of mathematical ones. This formalism has already been successfully used to address some challenges of computer science, for example to build a general framework for the specification of concurrent systems [14] or to allow the compositionality of machine learning components [46]. To the best of our knowledge, it has never been studied as foundation for a complete formal framework for data lakes. Nevertheless, some works tackle relevant issues to these systems with category theory.

Related to the data storage functionality, schemas and data instances in relational databases have been modeled with small categories and set-valued functors in [47], and constraints with functors and natural transformations later in [49]. Frameworks for object-oriented databases have been proposed in [30] and for document-oriented databases in [51]. The management of multi-model data and data integration issues are studied in [28, 32, 52] with some basic categorical tools like categories and functors as well as with more advanced one like pullbacks, pushforwards and kan lifts. Finally, metadata models based on category theory have been proposed in [7, 12].

On data maintenance, category theory has been mostly used to ensure data quality, through a metamodel adapted for geographic information systems in [18] and through a framework for variability models of software engineering in [35].

Finally, on data exploration, most of the existing works using category theory focus on query processing. In [22, 23], monads have been proposed as representation for queries, and monad comprehension is used for query processing. A query language implemented with the functional programming language Haskell, based on functors and using natural transformations for optimization is presented in [50]. Category theory also serves as basis for a framework of a search meta-engine in [53]. Other issues related to the data exploration fonctionality of data lakes include data and schema integration, which has been tackled in [48] with functorial data migration operations, and the creation of data visualization, which has been formalized in [54].

2.4 Synthesis

Data lakes have been detailed several times in terms of functionalities, but a formal definition expressed through a theoretical framework is still missing, and existing proposals either lack expressiveness or restrictiveness. Category theory is a promising candidate as foundation for such framework and has already been used in several relevant contexts for data lakes. A categorical definition unifying all the main functionalities of data lakes should provide the formal framework needed to improve them with mathematical properties and theorems, while creating a bridge with the previous works using the same formalism.

3 FORMALIZATION

In this section, after a brief introduction to category theory, we give a high level theoretical description of the main functionalities of a data lake presented in section 2.1. We show how composition and abstraction can be used to define different levels of representation and how different types of functors can ensure the navigation among them. We explain how our framework can be used to check the validity of an implementation of a data lake. We finally illustrate this on an example in subsection 3.3.

3.1 Category theory in a nutshell

Category Theory describes structures as categories and relations between them with functors.

A **category** C is defined by a collection $Ob(C)$ of **objects**, a collection $Hom(C)$ of directed relations between these objects, called **morphisms**, and a binary associative operation (noted $\circ$) to compose morphisms. The sub-collection of morphisms between an object x (called domain) and an object y (called codomain), both in $Ob(C)$, can be expressed as $Hom_C(x, y)$, and a morphism f between these objects is noted $f : x \rightarrow y$. Each object $x \in Ob(C)$ is associated with an identity morphism $id_x : x \rightarrow x$, acting as neutral element with $\circ$.

A category C is said to be **locally small** if $Hom(C)$ is a set, **small** if $Ob(C)$ and $Hom(C)$ are both sets and **large** otherwise. There is also a large category **Cat** defined with all objects as small categories and with all morphisms as functors between them.

A **functor** $F : C \rightarrow D$ is a structure mapping the objects and morphisms of a category C to objects and morphisms of a category D. Functors **preserve identities**, that is $\forall x \in Ob(C), F(id_x) = id_{F(x)}$, and **preserve composition**, that is $\forall f : x \rightarrow y, g : y \rightarrow z, F(g \circ f) = F(g) \circ F(f)$.

A **constant functor** $\Delta_{C-D} : C \rightarrow D$ is a special mapping that collapses every object in $Ob(C)$ to a single object $d \in Ob(D)$ and every morphism in $Hom(C)$ to the identity morphism id_d. **Surjective functors** act on every not empty $Hom_D(x, y)$. A functor $F : C \rightarrow D$ is said to be surjective if for every $x, y \in Ob(D)$ and every morphism in $Hom_D(x, y)$, there is at least one morphism in $Hom_C(F^{-1}(x), F^{-1}(y))$ (surjective mapping on every morphism of D).

A **product** of two categories $C1$ and $C2$ produces a new category whose objects are all the possible pairs (x, y) with $x \in \text{Ob}(C1)$ and $y \in \text{Ob}(C2)$ and morphisms $(x, y) \rightarrow (x', y')$ are pairs (f, g) where $f : x \rightarrow x' \in \text{Hom}_{C1}(x, x')$ and $g : y \rightarrow y' \in \text{Hom}_{C2}(y, y')$. A **bifunctor** has a product of categories as domain, and a category as codomain.

3.2 Categorical Framework for Data Lakes

In this subsection, we propose a high level categorical framework for data lakes, based on the description of the main functionalities established in the literature and presented in section 2.

At the highest level of abstraction, a data lake can be seen as a large category **DL**. This category is more precisely defined in table 1 and is visually represented in figure 1. Its collection of objects $Ob(\mathbf{DL})$ includes three of the main functionalities identified in section 2, namely Data Storage, Data Ingestion and Data Exploration. These objects are themselves categories. The Data Maintenance functionality has a specific representation. As it mainly transforms data to improve their usability, it is represented by a bifunctor **Storage** $\times$ **Storage** $\rightarrow$ **Storage**. It allows to define maintenance operations that can have a single dataset along with its metadata as input, but also operations that take two datasets with their metadata as input (such as the discovery of relationship between entities).

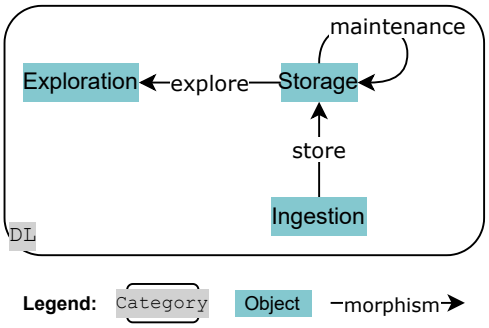

Figure 1: Representation of the category DL

The figure 2 and the tables 1 and 2 synthesize the following statements. The reader can use them as an helping guide of lecture.

The Data Ingestion functionality represents the entry point for data into the data lake. To do so, its **Ingestion** category contains three objects: *raw_data* for data as they are from their source, *dataset* for data as they enter into the data lake system and *metadata* that are extracted from *dataset*. The morphisms in this category show the different steps of data ingestion, i.e., we *load* the *raw_data* into the system to create a *dataset*, this *dataset* can be *transformed* (for example to compute aggregated data or to add some lightweight information such as the timestamp of the entry in the data lake), and some *metadata* can be *extracted* from this *dataset*.

The *dataset* and its *metadata* must be stored into the data lake. This functionality is supported by the **Storage** category. It is a category, in which objects represent data at different levels of abstraction, linked by morphisms. The different levels considered are the physical one with *md_system* and *d_system*, the logical one with *dataset*, the conceptual one with *metadata* and the structural one with *md_model* and *d_model*. The morphims link a *dataset* to its *metadata*, and the *dataset* and *metadata* to their respective *model*. The different *models* are themselves linked to their storage *system*.

Once the data are stored, they can be accessed in order to be explored. It can be done with the **Exploration** category. The *catalogue* allows the *localization* of a *dataset*, through its *metadata*. It is possible to run some *queries* or *algorithms* on the retrieved *dataset* in order to get the desired result.

With this representation, constant functors can link different levels when lower level categories are embedded in objects of higher level categories. For example, the refinements carried by the **Storage** category are all embedded in one object of **DL**, namely *Storage*. A constant functor $\Delta_{Storage-DL}$: **Storage** $\rightarrow$ **DL** can be used to link the lower level category to the higher level one. Functorial laws are satisfied because every object and every identity morphism in the domain category is mapped to the same object and its identity

Name	Objects	Morphisms (without identity morphisms)
DL	storage, ingestion, exploration	$store : Ingestion \rightarrow Storage$ $explore : Storage \rightarrow Exploration$ $maintenance : Storage \times Storage \rightarrow Storage$
Ingestion	raw_data, dataset, metadata	$load : raw\_data \rightarrow dataset$ $transform : dataset \rightarrow dataset$ $extract : dataset \rightarrow metadata$
Storage	metadata, dataset, d_model, d_system md_model, md_system	$described\_by : dataset \rightarrow metadata$ $md\_modeled\_by : metadata \rightarrow md\_model$ $md\_stored\_in : md\_model \rightarrow md\_system$ $d\_modeled\_by : dataset \rightarrow d\_model$ $d\_stored\_in : d\_model \rightarrow d\_system$
Exploration	catalogue, dataset metadata	$localize : catalogue \rightarrow dataset$ $query : dataset \rightarrow dataset$ $algorithm : dataset \rightarrow dataset$ $described\_by : dataset \rightarrow metadata$

Table 1: High level categories of the framework

Name	Type	Elements in Domain	Elements in Codomain
$store$	$Ingestion \rightarrow Storage$	raw_data	$dataset$
		$dataset$	$dataset$
		$metadata$	$metadata$
		$load$	$id_{dataset}$
		$transform$	$id_{dataset}$
		$extract$	$described\_by$
$explore$	$Storage \rightarrow Exploration$	$dataset$	$dataset$
		d_model	$dataset$
		d_system	$dataset$
		$metadata$	$metadata$
		md_model	$metadata$
		d_system	$metadata$
		$described\_by$	$described\_by$
		$d\_modeled\_by$	$id_{dataset}$
		d_stored_in	$id_{dataset}$
		$md\_modeled\_by$	$id_{metadata}$
		md_stored_in	$id_{metadata}$

Table 2: Functors used in the framework

morphism in the codomain category (identity preservation) and because identity morphisms can always be composed with themselves (composition preservation).

To use the categories of the framework with an instance of a data lake, each implementation of a functionality must be represented in a category, that must be linked to its higher-level corresponding category with a surjective functor. The surjective condition ensures the respect of the proposed framework and its structure, while allowing more complete descriptions of functionalities in the category of the implementation. Furthermore, the morphisms of the **DL** category add a constraint that forces the existence of functors between the categories concerned by each morphism when they are represented as objects in the **DL** category.

To ensure the **navigation between the functionalities of the data lake**, functors exist between the **Ingestion** and the **Storage** categories, and between the **Storage** and the **Exploration** categories. The defined functors and morphisms force the direction of the different transformations, and avoid the scattering of data. As the categories that will be defined for the implementation must be linked to their corresponding higher-level category, this forces also the implementation to provide these functors between the different implemented functionalities.

By relying on their constraints, such as the preservation of identities and the composition of morphisms, functors allow to **keep track of data** from their loading into the data lake to the production of results from a dataset in the exploration phase. Indeed, all the objects and the morphisms of the domain category must be sent to the codomain category. So, no information is lost when switching from a functionality to another.

The **maintenance** functionality takes a different form in the formalization framework. As it aims at improving the quality of the datasets or the metadata in order to ease their use during the exploration phase, two major applications of a maintenance operation can be found: improving a dataset or its metadata directly

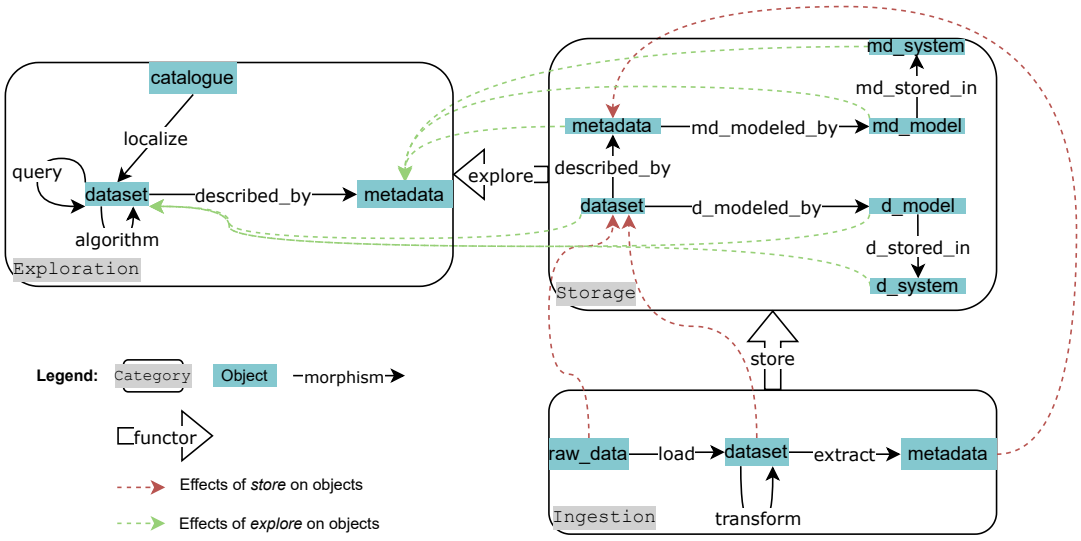

Figure 2: Representation of the categories Ingestion, Storage and Exploration, and the functors that link them

from themselves or improving a dataset or its metadata by relying on another dataset along with its metadata. Within the category theory, it is possible to use bifunctors to cover both of these applications. To do so, the bifunctor *maintenance* is defined as **Storage** × **Storage** → **Storage**. The first **Storage** category corresponds to the one on which the maintenance operation is applied, the second one corresponds to the one that will bring the elements required to the improvement (it can be the same **Storage** category as the first one when it is used to improve itself). The last **Storage** category is the result of the maintenance operation, and can be seen as the evolution of the first **Storage** category.

The figure 3 gives an overview of this mechanism. The objects (*dataset*, *metadata*) and (*metadata*, *dataset*) can be mapped either on the object *metadata* or on the object *dataset* of the resulting **Storage** category depending on the effect of the maintenance operation. We detail a little more this mechanism with an example in the following section. The other associations of objects are omitted for the sake of readability.

This representation **allows the composition of multiple maintenance operations**, while giving freedom to apply them in any order and to any data. Thus, the maintenance operations can be chosen according to the individual needs of each dataset. Furthermore, it has also constraints, as a **maintenance operation can only be applied on existing datasets and metadata**. This contributes to the identification of dependencies among datasets and metadata.

So, in its globality, this framework **keeps track of the entire lineage of data**. It is possible to know the source of data, how they have been transformed and the relationships that exist among datasets. It also imposes **constraints** on the validity of transformations applied on data, and on the switching from a functionality to another. Furthermore, **operations can be composed**, mainly with the *maintenance* functor, in order to allow a high flexibility that is essential to data lakes.

3.3 Example

We propose to show the usability of our categorical framework by defining a small data lake. We rely on the following use case: an enterprise has data about their customers, and records their online activity on the enterprise web applications.

The instances of the **Ingestion** and **Storage** categories are represented in figure 4. Regarding the data of the online activity, the enterprise is not interested by all the data, so it performs an aggregation operation in order to store only a summary of the data (category **Ing_ds1**). The data of the customers are easier to handle. As they are extracted from the enterprise information system, they do not need any transformation before being stored into the data lake (category **Ing_ds2**).

Once the data are ingested, they are stored in the data lake. For the activity data, the dataset is modeled as time series and InfluxDB is used as storage system. For the metadata, they are modeled as graphs in Neo4j (category **Str_ds1**). Regarding the customer data, the metadata are stored in the file system as JSON format, and the dataset in the relational PostgreSQL database (category **Str_ds2**).

The table 3 states the effects of functors on the objects and morphisms from the instance categories of the figure 4 to the corresponding high-level categories **Ingestion** and **Storage**. Thus, the ingestion and storage functionalities of the implemented data lake satisfy the requirements of the formalization including the surjectivity condition on functors.

Once the two datasets are stored, a maintenance operation can be applied on them (figure 5). In this operation, the dataset of the temporal activity is enriched with the data about customers (category **Str_ds1_v2**) in order to gain more information regarding their different characteristics, for example their country that can be used to make the typical hours of activity more precise. With this enriched dataset, an exploration is performed, first to reduce the dataset on a given period, and then to execute an anomaly detection algorithm to reveal fraudulent uses.

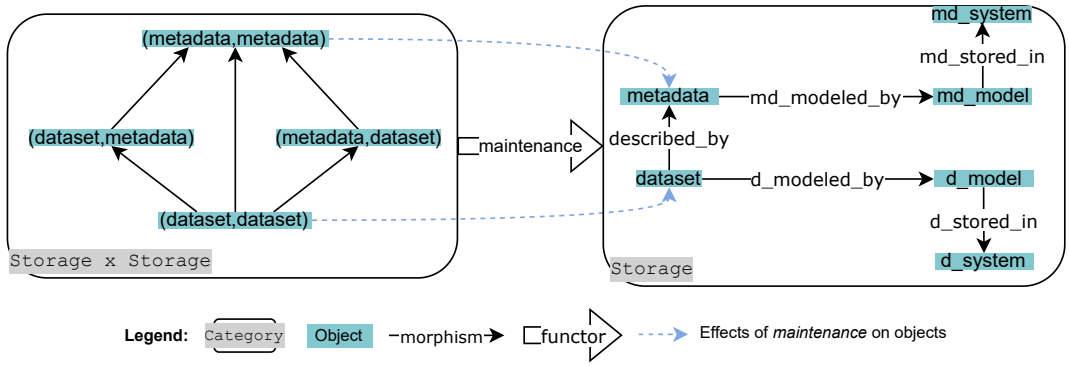

Figure 3: Partial representation of the *maintenance* functor, in which the immutable parts are represented

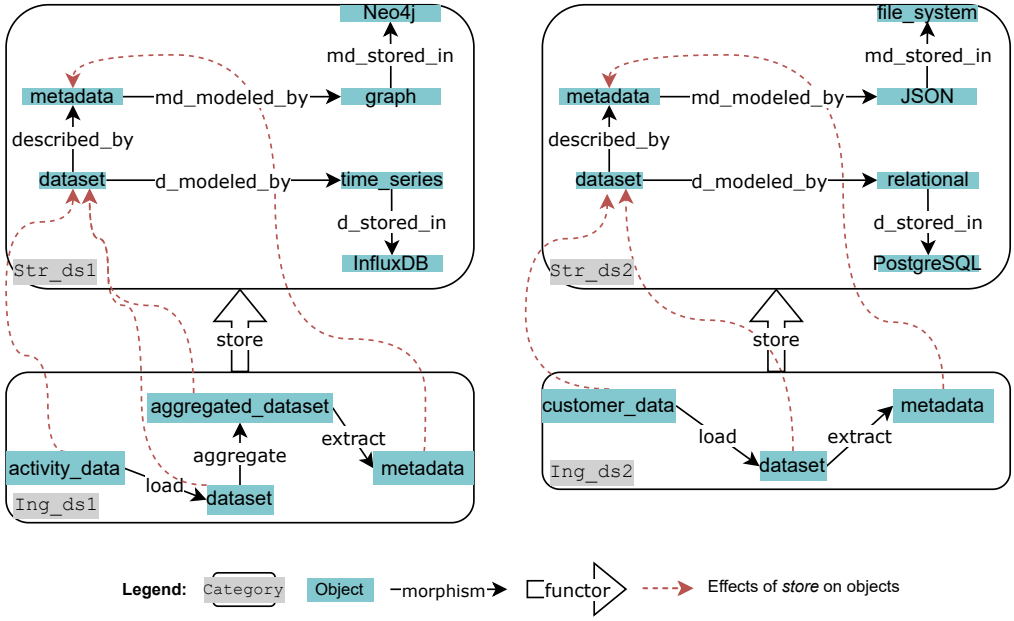

Figure 4: The ingestion and storage of the two datasets

This example demonstrates how the category theory supports the navigation across abstraction levels and how properties on functors constrain the implemented functionalities to comply to the structure defined in the corresponding higher-level categories. Moreover, the framework allows to represent all the potential functionalities of a data lake, and thus to use it to check the validity of an implementation of a data lake.

4 CONCLUSION

In this article, we have proposed a unified formal framework for data lakes based on category theory. The navigation between the different functionalities is controlled by functors and compositions, that allow to keep track of the lineage of data while providing the flexibility required by these systems. The levels of abstraction of the data lake are linked with constant or surjective functors, that ensure the validity of implementations of data lakes. We have shown on an example how the framework can be used.

Unlike previous works on the formalization of data lakes, our proposal considers a unified and complete view of all the main functionalities identified by the literature, namely Data Ingestion, Data Storage, Data Maintenance and Data Exploration. Thanks to the expressiveness and restrictiveness of category theory, we have also been able to represent and control the various dependencies and levels of abstraction existing in data lakes. Category theory additionally creates a bridge with existing works of the literature using the same formalism, allowing their use as refinements of some higher level aspects described in our framework.

As perspectives for future works, we plan to extend our formalism to allow the definition and control of complex and hybrid workflows for accessing and querying data in data lakes. Such workflows are indeed very important for these systems, in which a

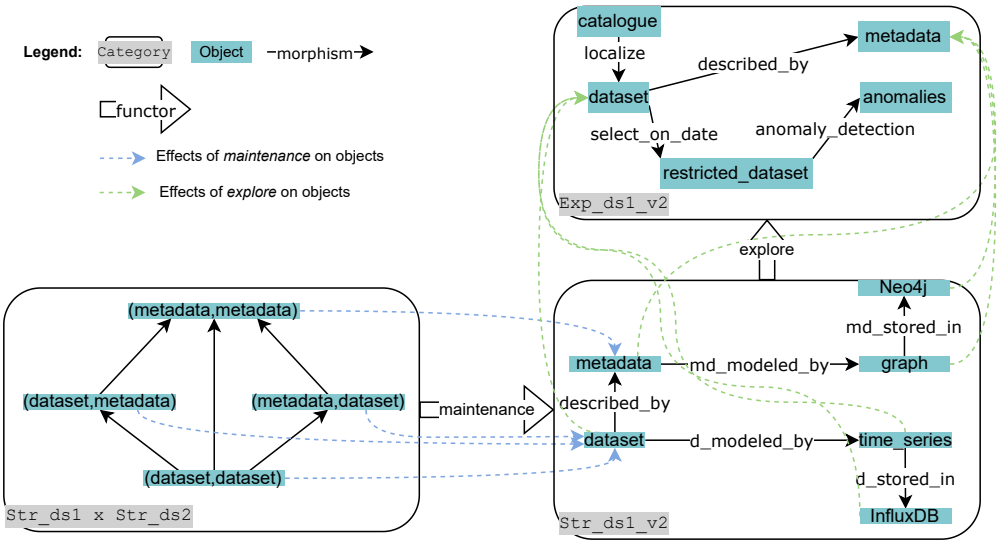

Figure 5: A maintenance operation followed by an exploration

variety of operations can be executed in the same environment like for example operators of relational algebra, machine learning tasks based on linear algebra, user-defined functions, etc. These various operations could therefore be unified by expressing them through categories, functors and bifunctors and then linked to the rest of the framework. We also plan to introduce the physical level of components in the data lake architecture by mapping the implemented functionalities to their corresponding component through functors. With this configuration, we can rely on a previous work [21] that allows to check the conservation or the loss of technical properties in an architecture with the category theory. We also think about exploring more the capabilities of the *maintenance* functor, that could be used to control the models of the dataset and the metadata depending on their original models with the (d_model, d_model) object of the product of categories. It can serve to detect model transformations that will lose precision compared to the original model.

REFERENCES

[1] Ayman Alserafi, Alberto Abelló, Oscar Romero, and Toon Calders. 2016. Towards information profiling: data lake content metadata management. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*. IEEE, 178–185.

[2] Michael Armbrust, Tathagata Das, Liwen Sun, Burak Yavuz, Shixiong Zhu, Mukul Murthy, Joseph Torres, Herman van Hovell, Adrian Ionescu, Alicja Łuszczak, et al. 2020. Delta lake: high-performance ACID table storage over cloud object stores. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3411–3424.

[3] Michael Armbrust, Ali Ghodsi, Reynold Xin, and Matei Zaharia. 2021. Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics. In *Proceedings of CIDR*.

[4] Amin Beheshti, Boualem Benatallah, Reza Nouri, and Alireza Tabebordbar. 2018. CoreKG: a knowledge lake service. *Proceedings of the VLDB Endowment* 11, 12 (2018), 1942–1945.

[5] Manfred Broy. 2011. Can practitioners neglect theory and theoreticians neglect practice? *Computer* 44, 10 (2011), 19–24.

[6] Manfred Broy and María Victoria Cengarle. 2011. UML formal semantics: lessons learned. *Software & Systems Modeling* 10, 4 (2011), 441–446.

[7] Isabel Cafezeiro and Edward Hermann Haeusler. 2007. Semantic Interoperability via Category Theory.. In *ER (Tutorials, Posters, Panels & Industrial Contributions)*. Citeseer, 197–202.

[8] Edgar Frank Codd. 1983. A relational model of data for large shared data banks. *Commun. ACM* 26, 1 (1983), 64–69.

[9] Julia Couto, Olimar Teixeira Borges, Duncan D Ruiz, Sabrina Marczak, and Rafael Prikladnicki. 2019. A Mapping Study about Data Lakes: An Improved Definition and Possible Architectures.. In *SEKE*. 453–578.

[10] Zhamak Dehghani. 2019. How to move beyond a monolithic data lake to a distributed data mesh. *Martin Fowler's Blog* (2019).

[11] Claudia Diamantini, Domenico Potena, and Emanuele Storti. 2021. A Semantic Data Lake Model for Analytic Query-Driven Discovery. In *The 23rd International Conference on Information Integration and Web Intelligence*. 183–186.

[12] Zinovy Diskin. 1997. The Arrow Logic of Metadata Environment: A Formalised Graph-Based Framework for Structuring Metadata Repositories. (1997).

[13] James Dixon. 2010. Pentaho, Hadoop, and data lakes. *blog, Oct* (2010).

[14] Hartmut Ehrig, Martin Große-Rhode, and Uwe Wolter. 1998. Applications of category theory to the area of algebraic specification in computer science. *Applied categorical structures* 6, 1 (1998), 1–35.

[15] Samuel Eilenberg and Saunders MacLane. 1945. General theory of natural equivalences. *Trans. Amer. Math. Soc.* 58, 2 (1945), 231–294.

[16] Mina Farid, Alexandra Roatis, Ihab F Ilyas, Hella-Franziska Hoffmann, and Xu Chu. 2016. CLAMS: bringing quality to data lakes. In *Proceedings of the 2016 International Conference on Management of Data*. 2089–2092.

[17] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1001–1012.

[18] Andrew U Frank. 1998. Metamodels for data quality description. *Data Quality in Geographic Information-From Error to Uncertainty* 192 (1998).

[19] Yihan Gao, Silu Huang, and Aditya Parameswaran. 2018. Navigating the data lake with datamaran: Automatically extracting structure from log datasets. In *Proceedings of the 2018 International Conference on Management of Data*. 943–958.

[20] I. Gartner. 2014. Gartner Says Beware of the Data Lake Fallacy. https://www.gartner.com/newsroom/id/2809117.

[21] Annabelle Gillet, Éric Leclercq, and Nadine Cullot. 2021. Lambda+, the Renewal of the Lambda Architecture: Category Theory to the Rescue. In *International Conference on Advanced Information Systems Engineering*. Springer, 381–396.

[22] Georg Gottlob and Christoph Koch. 2002. Monadic queries over tree-structured data. In *Proceedings 17th annual IEEE symposium on logic in computer science*. IEEE, 189–202.

[23] Torsten Grust. 2004. Monad comprehensions: a versatile representation for queries. In *The Functional Approach to Data Management*. Springer, 288–311.

[24] Rihan Hai, Sandra Geisler, and Christoph Quix. 2016. Constance: An intelligent data lake system. In *Proceedings of the 2016 international conference on management of data*. 2097–2100.

[25] Rihan Hai, Christoph Quix, and Matthias Jarke. 2021. Data lake concept and systems: a survey. *arXiv preprint arXiv:2106.09592* (2021).

[26] Rihan Hai, Christoph Quix, and Dan Wang. 2019. Relaxed functional dependency discovery in heterogeneous data lakes. In *International Conference on Conceptual Modeling*. Springer, 225–239.

Functor	Element in domain	Element in codomain
$Ing\_ds1 \rightarrow Ingestion$	$activity\_data$	raw_data
	$dataset$	$dataset$
	$aggregated\_dataset$	$dataset$
	$metadata$	$metadata$
	$load$	$load$
	$aggregate$	$transform$
	$extract$	$extract$
$Ing\_ds2 \rightarrow Ingestion$	$customer\_data$	raw_data
	$dataset$	$dataset$
	$metadata$	$metadata$
	$load$	$load$
	$id_{dataset}$	$transform$
	$extract$	$extract$
$Str\_ds1 \rightarrow Storage$	$dataset$	$dataset$
	$time\_series$	d_model
	$InfluxDB$	d_system
	$metadata$	$metadata$
	$graph$	md_model
	$Neo4j$	d_system
	$described\_by$	$described\_by$
	$d\_modeled\_by$	$d\_modeled\_by$
	d_stored_in	d_stored_in
	$md\_modeled\_by$	$md\_modeled\_by$
	md_stored_in	md_stored_in
$Str\_ds2 \rightarrow Storage$	$dataset$	$dataset$
	$relational$	d_model
	$PostgreSQL$	d_system
	$metadata$	$metadata$
	$JSON$	md_model
	$file\_system$	d_system
	$described\_by$	$described\_by$
	$d\_modeled\_by$	$d\_modeled\_by$
	d_stored_in	d_stored_in
	$md\_modeled\_by$	$md\_modeled\_by$
	md_stored_in	md_stored_in

Table 3: Functors used in the example, between the category representing the instance of a functionality and the high-level category of the functionality

[27] Alon Y Halevy, Flip Korn, Natalya Fridman Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. 2016. Managing Google's data lake: an overview of the Goods system. *IEEE Data Eng. Bull.* 39, 3 (2016), 5–14.

[28] Irena Holubova, Pavel Contos, and Martin Svoboda. 2021. Categorical Management of Multi-Model Data. In *25th International Database Engineering & Applications Symposium*. 134–140.

[29] Pontus Johnson, Mathias Ekstedt, and Ivar Jacobson. 2012. Where's the theory for software engineering? *IEEE software* 29, 5 (2012), 96–96.

[30] P Kolencık. 1998. *Categorical Framework for Object-Oriented Database Model*. Ph. D. Dissertation. PhD thesis.

[31] Pengfei Liu, Sabine Loudcher, Jérôme Darmont, and Camille Noûs. 2021. ArchaeoDAL: A Data Lake for Archaeological Data Management and Analytics. In *25th International Database Engineering & Applications Symposium*. 252–262.

[32] Zhen Hua Liu, Jiaheng Lu, Dieter Gawlick, Heli Helskyaho, Gregory Pogossiants, and Zhe Wu. 2018. Multi-model database management systems-a look forward. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Springer, 16–29.

[33] Antonio Maccioni and Riccardo Torlone. 2018. KAYAK: a framework for just-in-time data preparation in a data lake. In *International Conference on Advanced Information Systems Engineering*. Springer, 474–489.

[34] Jacob McPadden, Thomas JS Durant, Dustin R Bunch, Andreas Coppi, Nathan Price, Kris Rodgerson, Charles J Torre Jr, William Byron, H Patrick Young, Allen L Hsiao, et al. 2018. A scalable data science platform for healthcare and precision medicine research. *arXiv preprint arXiv:1808.04849* (2018).

[35] Daniel-Jesus Munoz, Dilian Gurov, Monica Pinto, and Lidia Fuentes. 2021. Category Theory Framework for Variability Models with Non-functional Requirements. In *International Conference on Advanced Information Systems Engineering*. Springer, 397–413.

[36] Amr A Munshi and Yasser Abdel-Rady I Mohamed. 2018. Data lake lambda architecture for smart grids big data analytics. *IEEE Access* 6 (2018), 40463–40471.

[37] Iuri D Nogueira, Maram Romdhane, and Jérôme Darmont. 2018. Modeling data lake metadata with a data vault. In *Proceedings of the 22nd International Database Engineering & Applications Symposium*. 253–261.

[38] Christoph Quix, Rihan Hai, and Ivan Vatov. 2016. Metadata extraction and management in data lakes with GEMMS. *Complex Systems Informatics and Modeling Quarterly* 9 (2016), 67–83.

[39] Raghu Ramakrishnan, Baskar Sridharan, John R Douceur, Pavan Kasturi, Balaji Krishnamachari-Sampath, Karthick Krishnamoorthy, Peng Li, Mitica Manu, Spiro Michaylov, Rogério Ramos, et al. 2017. Azure data lake store: a hyperscale distributed file service for big data analytics. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 51–63.

[40] Sarathkumar Rangarajan, Huai Liu, Hua Wang, and Chuan-Long Wang. 2015. Scalable architecture for personalized healthcare service recommendation using big data lake. In *Service research and innovation*. Springer, 65–79.

[41] Franck Ravat and Yan Zhao. 2019. Data lakes: Trends and perspectives. In *International Conference on Database and Expert Systems Applications*. Springer, 304–313.

[42] Franck Ravat and Yan Zhao. 2019. Metadata management for data lakes. In *European Conference on Advances in Databases and Information Systems*. Springer, 37–44.

[43] David Sarramia, Alexandre Claude, Francis Ogereau, Jérémy Mezhoud, and Gilles Mailhot. 2022. CEBA: A Data Lake for Data Sharing and Environmental Monitoring. *Sensors* 22, 7 (2022), 2733.

[44] Pegdwendé Sawadogo and Jérôme Darmont. 2021. On data lake architectures and metadata management. *Journal of Intelligent Information Systems* 56, 1 (2021), 97–120.

[45] Etienne Scholly, Pegdwendé Sawadogo, Pengfei Liu, Javier Alfonso Espinosa-Oviedo, Cécile Favre, Sabine Loudcher, Jérôme Darmont, and Camille Noûs. 2021. Coining goldMEDAL: a new contribution to data lake generic metadata modeling. *arXiv preprint arXiv:2103.13155* (2021).

[46] Dan Shiebler, Bruno Gavranović, and Paul Wilson. 2021. Category theory in machine learning. *arXiv preprint arXiv:2106.07032* (2021).

[47] David Spivak. 2011. *Categorical Information Theory*. Technical Report. Massachusetts Inst. of Tech.

[48] David I Spivak. 2012. Functorial data migration. *Information and Computation* 217 (2012), 31–51.

[49] David I Spivak. 2014. Database queries and constraints via lifting problems. *Mathematical structures in computer science* 24, 6 (2014).

[50] Laurent Thiry, Heng Zhao, and Michel Hassenforder. 2018. Categories for (Big) Data models and optimization. *Journal of Big Data* 5, 1 (2018), 1–20.

[51] David Toth. 2008. Database engineering from the category theory viewpoint. *Databases, Texts* (2008), 37.

[52] Valter Uotila and Jiaheng Lu. 2021. A Formal Category Theoretical Framework for Multi-model Data Transformations. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Springer, 14–28.

[53] Paul-Christophe Varoutas, Philippe Rizand, and Alain Livartowski. 2006. Using category theory as a basis for a heterogeneous data source search meta-engine: the Prométhée framework. In *International Conference on Algebraic Methodology and Software Technology*. Springer, 381–387.

[54] Paul Vickers, Joe Faith, and Nick Rossiter. 2012. Understanding visualization: A formal approach using category theory and semiotics. *IEEE transactions on visualization and computer graphics* 19, 6 (2012), 1048–1061.

[55] Yan Zhao, Imen Megdiche, Franck Ravat, and Vincent-nam Dang. 2021. A Zone-Based Data Lake Architecture for IoT, Small and Big Data. In *25th International Database Engineering & Applications Symposium*. 94–102.