HAL open science

Modeling Data Lake Metadata with a Data Vault

Iuri Nogueira, Maram Romdhane, Jérôme Darmont

▶ To cite this version:

Iuri Nogueira, Maram Romdhane, Jérôme Darmont. Modeling Data Lake Metadata with a Data Vault. 22nd International Database Engineering & Applications Symposium (IDEAS 2018), Jun 2018, Villa San Giovanni, Italy. pp.253-261. hal-01788036

HAL Id: hal-01788036

https://hal.science/hal-01788036v1

Submitted on 10 Jul 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Modeling Data Lake Metadata with a Data Vault

Iuri D. Nogueira
Université de Lyon, Lyon 2
France
iuri.deolindonogueira@univ-lyon2.fr

Maram Romdhane
Université de Lyon, Lyon 2
France
maram.romdhane@univ-lyon2.fr

Jérôme Darmont
Université de Lyon, Lyon 2, ERIC EA 3083
5 avenue Pierre Mendès France, F69676 Bron Cedex, France
jerome.darmont@univ-lyon2.fr

Abstract

With the rise of big data, business intelligence had to find solutions for managing even greater data volumes and variety than in data warehouses, which proved ill-adapted. Data lakes answer these needs from a storage point of view, but require managing adequate metadata to guarantee an efficient access to data. Starting from a multidimensional metadata model designed for an industrial heritage data lake presenting a lack of schema evolutivity, we propose in this paper to use ensemble modeling, and more precisely a data vault, to address this issue. To illustrate the feasibility of this approach, we instantiate our metadata conceptual model into relational and document-oriented logical and physical models, respectively. We also compare the physical models in terms of metadata storage and query response time.

Keywords: Big data, Data lake, Metadata management, Ensemble modeling, Data vault

Topics: Information systems, Information storage systems, Cloud based storage

1 Introduction

Born with big data in the 2010's, data lakes propose a way to store in their native format voluminous and diversely structured data, in an evolutionary storage place allowing later analyses (reporting, visualization, data mining...) [3]. The concept of data lake opposes that

of data warehousing, which outputs an integrated, highly structured, decision-oriented and subject-oriented database, but has the disadvantage of dividing data into silos [26].

Yet, everyone agrees that a data lake must be well designed. Otherwise, it is doomed to quickly become an inoperable data swamp [1, 9]. That is, a data lake must allow querying data (selection/restriction) with good response times, and not only storing data and handling a "key-value" access. However, actual solutions are more or less non-existent in the literature and pertain to undisclosed industrial practices.

This is why [21] proposed a conceptual metadata model that allows indexing and efficiently querying an industrial heritage data lake constituted of XML data, texts, floorplans and pictures. This multidimensional model is similar to snowflake models used in data warehouses or datamarts, but only stores metadata, not the document corpus itself.

This metadata model has been instantiated physically in several NoSQL database management systems (DBMSs) to enable scaling. However, such a data warehouse-like schema cannot evolve easily when data sources evolve themselves, or when new sources pop up, whereas this is a crucial point in the management of a data lake.

Consequently, we propose in this paper:

1. to replace the multidimensional model of [21] by an ensemble model, more precisely a data vault [16, 8, 4, 18], which is a data model allowing easy schema evolutions and has, to the best of our knowledge, never been used in the context of metadata management;

2. to evaluate, the feasibility, on one hand, and the effectiveness of this model in terms of metadata query response (as an index), on the other hand, because it induces many joints. For this sake, we translate our conceptual metadata vault into different logical (i.e., relational and document-oriented) and physical (i.e., PostgreSQL[1] and MongoDB[2]) models, which also helps compare the respective efficiency of the two physical models.

The remainder of this paper is organized as follows. Section 2 is devoted to a state of the art concerning data lakes and metadata management, on one hand, and ensemble modeling and data vaults in particular, on the other hand. Section 3 presents our conceptual metadata vault model, as well as its logical and physical translations. Section 4 details the experiments we conducted to validate the feasibility of our approach and compare the PostgreSQL and MongoDB physical models in terms of storage and metadata query response time. Finally, Section 5 concludes this paper and hints at research perspectives.

[1] `https://www.postgresql.org`
[2] `https://www.mongodb.com`

2 Related Work

2.1 Data Lakes and Metadata

Data lakes aim at quickly integrate very large volumes of mixed types data, i.e., from structured to unstructured data [3, 20, 9]. However, they are not only a storage technology, i.e., mostly the Hadoop Distributed File System (HDFS), but offer a new data ecosystem that allows cross-analyzing data on demand [6], without needing costly preprocessing tasks as in data warehousing processes. Data are immediately accessible through the schema-on-read (or late binding) approach [5], unlike, again, in data warehouses that are periodically refreshed through an Extraction, Transformation and Loading (ETL) phase (schema-on-write) that may be expensive [19].

In summary, data lakes manage data with variety for on-demand, ad-hoc analyses; while data warehouses manage structured data for recurring, industrialized analyses.

Data lakes are typically subdivided into three components [26, 1]:

1. a storage system: very often HDFS, though NoSQL DBMSs are also an option [9, 14];
2. a metadata system, which we focus on below;
3. an access and analysis system generally relying on MapReduce or Spark [10].

By contrast, [9] proposes an architecture with respect to data types, made of so-called data ponds for raw data, analog (stream) data, application data (i.e., a data warehouse), textual data and archived data, respectively. In such a context, database research on querying multi-store systems [2, 27] looks highly relevant.

Data lakes mostly bear a "flat" architecture, where each data element has a unique identifier and a set of characterizing tags [19]. Such tags are actually essential metadata to help comprehend data and access data [1], not to mention query effectiveness.

The advantage of data tagging is that new data and new sources can be inserted on the fly. Once data are tagged, they just need to be connected to already stored data. This feature allows users to formulate queries directly, without needing the help of a business intelligence expert. Yet, this feature requires that the metadata system can manage this evolutive data structure.

2.2 Ensemble Modeling and Data Vaults

Ensemble modeling is an approach dating back from the early 2010's, which aims to renormalize data warehouses to bring them closer to the business concepts they model; and to allow better evolutivity, both in terms of data and schema [24].

The two prominent approaches in ensemble modeling are data vaults [16, 8, 4, 18] and anchor modeling [23, 25]. Both are actually very close [24], with a somewhat easier evolutivity and a non-destructive schema evolution for anchor modeling, but a larger number of objects to

manage because of a sixth normal form (6NF) model, as well as non-automated maintenance procedures for timestamps. Data vaults are closer to traditional multidimensional models (they adopt the third normal form – 3NF) and are supported by a greater number of tools, which made us select data vaults in this work. However, this choice is not definitively settled.

Data vaults were invented to meet industrial needs [16]. A data vault is defined at the relational, logical level as a linked set of normalized, detail-oriented and history-tracking tables that support one or more functional domains in an organization. It is a hybrid approach encompassing 3NF and the star schema [16, 8, 4, 18].

This architecture allows incremental low-cost schema construction (making easy to modify business rules), seamless integration of unstructured data and loading in near real time. Data vaults also allow petabyte-scale big data management [18]. Finally, artificial intelligence and data mining methods can easily and directly be applied to a data vault [17].

More concretely, a data vault consists of the following main elements [16, 8, 4, 18].

- A *hub* is a basic entity that represents a business concept (a dimension hierarchy level in a classical data warehouse, e.g., customer or product). A hub mainly contains a business key.
- A *link* materializes an association between two or more hubs. It would correspond to a fact entity in a classical data warehouse.
- A *satellite* contains the various descriptive data related to a hub or link.

Finally, note that, although they are defined at the logical level by [16], these concepts can easily be transposed to the conceptual [11] and physical [15] levels. An example of conceptual data vault model is shown in Figure 1.

3 Metadata Vault Model

3.1 Data Corpus

To illustrate our proposal with a use case that can serve as a proof of concept, we exploit the data corpus constituted by the TECTONIQ project, which aims to highlight the textile industrial heritage of the Lille Metropolis in northern France [13, 12].

This corpus gathers heterogeneous data, supplied by different sources (Table 1) and must allow various types of analyses. Thence, the corpus is stored in a data lake whose metadata are multidimensionally modeled [21]. We propose in this paper to make these metadata evolutive when new data sources pop up.

3.2 Conceptual Model

Although the multidimensional metadata model by [21] is too large to be reproduced here, let us describe it briefly. It is designed around a set of dimensions with hierarchies: Date/Epoch,

Table 1: TECTONIQ corpus features

Source	**Inventory**	**La Voix du Nord**
Description	Industrial building descriptions	Press articles related to the textile industry
Number of documents	49	1
Number of instances	49	30
Overall size	120 KB	92 KB
Format	XML (data)	XML (documents)
Source	**IHRIS Pictures**	**Books**
Description	Photos and plans of buildings and monuments related to the textile industry	Public domain French history books
Number of documents	30	165
Number of instances	30	165
Overall size	2.78 MB	1011.25 MB
Format	JPEG	PDF

Keyword, Location/District/City, Source, Reference/Author; on which "fact" entities (without measures) that correspond to the data sources from Table 1 plug in.

Figure 1 illustrates our conceptual metadata vault model, which exploits the conceptual modeling of data vaults' logical, relational concepts [11]. Blue rounded rectangles represent hubs, gray rectangles satellites and the green hexagon a link. Associations between hubs and links are of cardinality "many to many" to ensure the greatest generality, while associations betweens hubs or links and satellites are of cardinality "one to many".

To identify hubs, which are the most important and most used entities in queries, we target the metadata related to the characteristics of each input document, which are most susceptible to constitute keys. Thus, we select title (*Hub_Title*), location (*Hub_Location*), date (*Hub_Date*) and document category (*Hub_Category*).

Then, we associate with each of these hubs a satellite that contains the hub's descriptive attributes. For example, the satellite connected to *Hub_Title*, *Sat_Title*, contains the *Title*, *Authors*, *Description* and *Keywords* attributes.

Hub_Category is associated with four satellites that form a classification corresponding to each of the data sources we have. The descriptive attributes of these satellites are specific to each source.

Link_Document allows to associate all hubs.

Finally, since our model concerns only metadata, each entity (hub, link, satellite) is described by a direct reference (*Source*) to a document (physical file) in the data lake. All data access are made through this reference.

Thanks to ensemble modeling, any new data source or schema evolution within the data corpus can be supported by adding satellites, in the simplest cases, or even hubs and links. In addition, entities that would become obsolete are simply identified by their timestamp

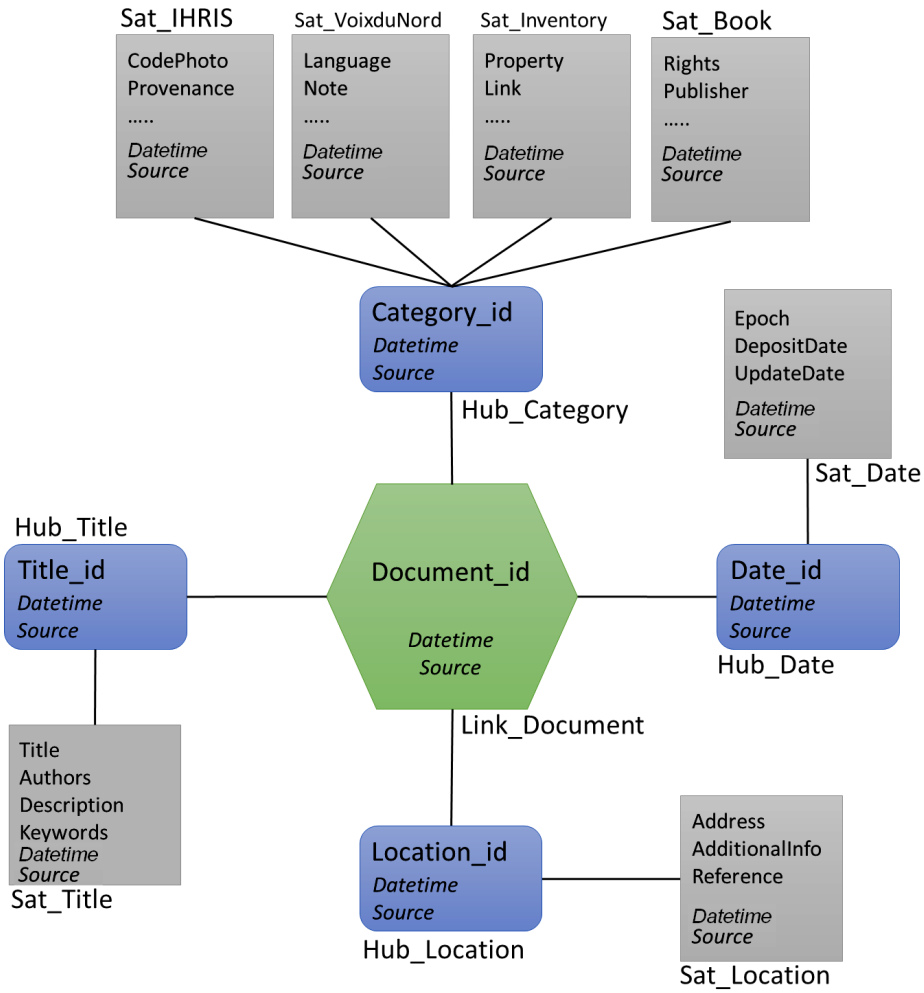

Figure 1: Metadata vault conceptual model

(*Datetime*).

3.3 Logical and Physical Models

To show that our conceptual model can adapt to different contexts, and to compare them, we translate it into two types of logical and physical models: a relational model with PostgreSQL and a document-oriented NoSQL model with MongoDB.

The relational metadata model (Table 2)[3] is classically translated from the conceptual model (Figure 1). Let us only note that, in Table 2, the elements of the attributes column that are postfixed by *_id* are foreign keys, and that satellite primary keys are also foreign keys that reference the corresponding hub's primary key. Eventually, the *Datetime* and *Source* attributes featured in all tables are not repeated for clarity.

[3]We adopt this unusual table presentation to have a uniform representation with the document-oriented model (Table 3) and facilitate the comparison of the two logical models.

Table 2: Metadata vault relational logical model

Table	**Primary key**	**Attributes**
Hubs		
Hub_Title	Title_id	
Hub_Date	Date_id	
Hub_Location	Location_id	
Hub_Category	Category_id	
Link		
Link_Document	Document_id	Title_id, Location_id, Date_id, Category_id
Satellites		
Sat_Title	Title_id, Datetime	Title, Authors, Description, Keywords
Sat_Date	Date_id, Datetime	Epoch, DepositDate, UpdateDate
Sat_Location	Location_id, Datetime	Address, AdditionalInfo, Reference
Sat_IRHIS	Category_id, Datetime	CodePhoto, Provenance
Sat_VoixDuNord	Category_id, Datetime	Language, Note
Sat_Inventory	Category_id, Datetime	Property, Link
Sat_Book	Category_id, Datetime	Rights, Publisher

Table 3: Metadata vault document-oriented logical model

Collection	**ObjectID**	**Fields**
Hubs		
Hub_Title	Title_id	
Hub_Date	Date_id	
Hub_Location	Location_id	
Hub_Category	Category_id	
Link		
Link_Document	Document_id	
Satellites		
Sat_Title	Title_id	Title, Authors, Description, Keywords
Sat_Date	Date_id	Epoch, DepositDate, UpdateDate
Sat_Location	Location_id	Address, AdditionalInfo, Reference
Sat_IRHIS	Category_id	CodePhoto, Provenance
Sat_VoixDuNord	Category_id	Language, Note
Sat_Inventory	Category_id	Property, Link
Sat_Book	Category_id	Rights, Publisher

The document-oriented metadata model (Table 3) exploits the notion of collection. Each hub, link and satellite is translated into a collection comprising various documents, each having an identifier (*ObjectID*). In the MongoDB physical model, this identifier is generated automatically when the document is created. As in the relational case, the *Datetime* and *Source* fields present in all collections are not repeated for clarity.

Each hub is associated with both *Link_Document* and its satellite(s) through its *ObjectID*. For example, *Hub_Title* is associated with *Link_Document* and *Sat_Title* through *Title_id*. Finally, Figure 2 presents an example of translation from the document-oriented logical model

into the MongoDB physical model in JavaScript Object Notation (JSON) format.


```
{
 "Sat_Book": {
  "00000C842_001": {
   "dc:relation": "http://www.sudoc.fr/122661389",
   "dc:creator": [
    "Petit, Jules",
    "Chambre de commerce et d'industrie (Boulogne-sur-Mer, Pas-de-Calais).",
    "Commission du projet de chemin de fer direct de Calais a Marseille"
   ],
   "dc:subject": "Calais-Marseille, Chemin de fer (France).",
   "dc:publisher": "Villeneuve d'Ascq : SCD Lille 3",
   "dc:date": "[2008]",
   "dc:format": "application/pdf",
   "dc:language": "fre",
   "dc:rights": "domaine public",
   "dc:title": "Projet de chemin de fer de Calais a Marseille rapport
    fait a la Chambre de commerce de Boulogne, le 25 novembre 1881",
   "dc:type": "text",
   "dc:source": "Bibliotheque Georges Lefebvre"
  }
 }
}
```


Figure 2: JSON excerpt from the *Sat_Book* collection of the MongoDB physical model

At the physical level, metadata are extracted from source documents through a script-based ETL process.

4 Experimental Validation

The objective of this section is to show the feasibility of our metadata vault model and to compare the physical models of Section 3.3 in terms of storage space and query response time.

We conduct our experiments on an Intel Core i5-5300U 2.30 GHz PC with 8 GB of memory, running Windows 64-bit. The DBMSs we use are PostgreSQL 9.6.2-1 and MongoDB ssl 3.4.3. The data corpus in this test set is the one presented in Table 1.

4.1 Storage Volume

After inserting metadata in the chosen DBMSs' native format, we measure the storage volume required by PostgreSQL and MongoDB (Table 4). Our measurements take into account indexes automatically created by the DBMSs (e.g., on primary keys in PostgreSQL; no other index is created), unused memory space, and the space that is released when deleting or moving data.

Table 4: Metadata volume (KB)

	PostgreSQL	**MongoDB**
Hub_Title	80	45
Hub_Date	64	36
Hub_Location	72	49
Hub_Category	64	36
Link_Document	48	36
Sat_Title	168	69
Sat_Date	48	36
Sat_Location	56	49
Sat_IHRIS	16	16
Sat_VoixDuNord	16	7
Sat_Inventory	88	45
Sat_Book	56	37
Total	**776**	**461**

We first note that the volume of metadata generated for 245 files is small (less than 1 MB) in both cases. In addition, MongoDB systematically requires less space than PostgreSQL. This is explained by the use of the JSON format, which is very light, in MongoDB. In contrast, in PostgreSQL, each table is stored as a page vector of predetermined size (8 KB), resulting in pages that are not fully filled.

4.2 Response Time

To measure the performance of the proposed metadata model, we formulate five queries of increasing complexity with respect to the number of hubs involved (and, therefore, to joins via *Link_Document*), which we apply on both physical models.

1. Retrieve all documents whose title contains the word "factory" (data sources: *Hub_Title*, *Sat_Title*).

2. Retrieve all documents whose title contains the word "factory" and whose address is "Tourcoing" (data sources: *Hub_ Title*, *Sat_Title*, *Hub_Location*, *Sat_Location*, *Link_Document*).

3. Retrieve all documents whose title contains the word "factory", whose address is "Tourcoing" and whose deposit date is in 2010 (data sources: *Hub_Title*, *Sat_Title*, *Hub_Location*, *Sat_Location*, *Hub_Date*, *Sat_Date*, *Link_Document*).

4. Retrieve all documents whose title contains the word "factory", whose address is "Tourcoing", whose deposit date is in 2010 and that belong to the "book" category (data sources: *Hub_Title*, *Sat_Title*, *Hub_Location*, *Sat_Location*, *Hub_Date*, *Sat_Date*, *Hub_Category*, *Sat_Book*, *Link_Document*).

5. Retrieve all titles containing the word "factory", and then the corresponding documents, whatever their category (data sources: *Hub_Title*, *Sat_Title*, *Hub_Category*, *Sat_IHRIS* and/or *Sat_VoixDuNord* and/or *Sat_Inventory* and/or *Sat_Book*, *Link_ Document*).

The translation of all queries in SQL and MongoDB's query language are provided in Appendices A and B, respectively.

Figure 3 features the average execution time of queries #1 to #4 under PostgreSQL and MongoDB, respectively. We execute each query 100 times (in the order indicated above) to compensate for any variation in execution time. Figure 3 shows tremendously low response times for MongoDB, while those obtained with PostgreSQL seem to evolve exponentially.

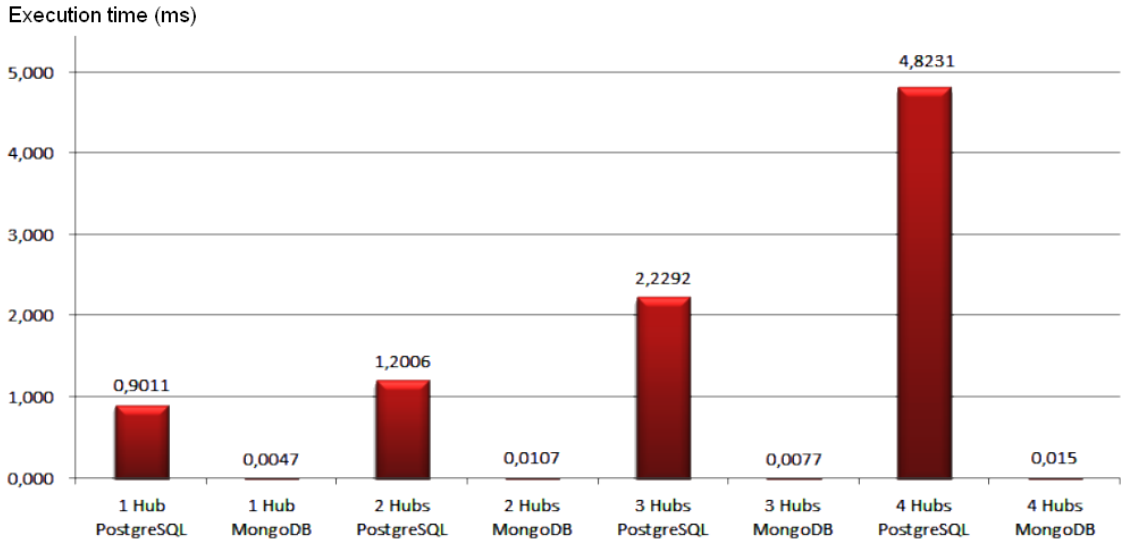

Figure 3: Average response times of queries #1 to #4

Figure 4 illustrates the average execution time of query #5, still for 100 executions of the query. It is distinct from Figure 3 because query #5 is more complex than the previous four. It is indeed not possible to dynamically determine the category satellite to use. It is therefore necessary to execute query #5 in two steps (subqueries): the first to determine the category and the second to retrieve the remaining information knowing the category. These two operations induce much longer response times than those of queries #1 to #4.

MongoDB is once again more efficient (three times faster on average) than PostgreSQL. This difference in query execution time can be explained by the internal query rewriting done by PostgreSQL, which creates a subquery inducing an expensive join [7], while MongoDB uses a search method that allows the rapid union of collections.

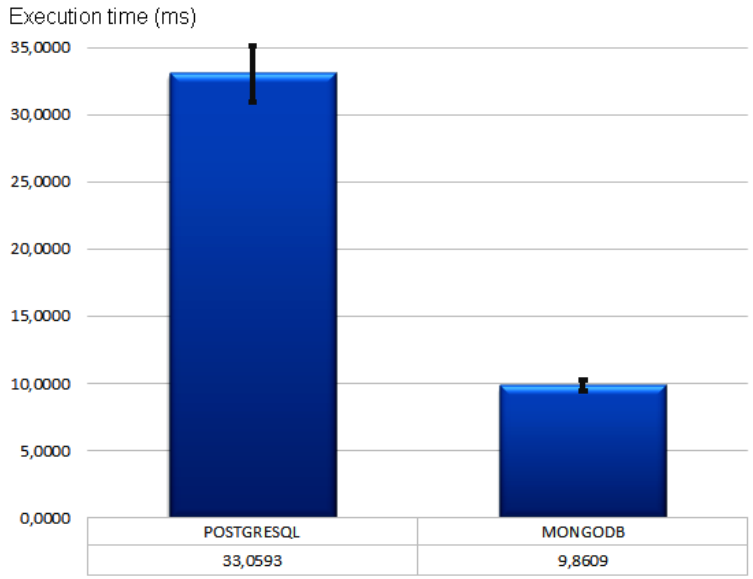

Figure 4: Average response time of query #5

5 Conclusion

From the modeling of data lake metadata as a multidimensional schema, noticing that schema evolution is not guaranteed, we instead propose in this paper an ensemble model, more precisely a metadata vault.

The translation of our conceptual metadata model into logical and physical models, as well as experiments carried out on the TECTONIQ corpus, help demonstrate the feasibility of our modeling choice in terms of metadata storage volume and response time to queries formulated on the metadata.

The comparison of two physical models (PostgreSQL and MongoDB) also reveals the superiority of document-based models for storing this type of metadata.

The perspectives that this preliminary work is opening are numerous. In particular, it would be necessary to test the robustness of the metadata model when source data scale up, add data sources to verify the relevance of data vault modeling and test the model with queries more complex than projections/restrictions.

Moreover, although the supporters of anchor modeling argue that 6NF modeling allows good response times thanks to the join eliminating technique used in modern optimizers [22], it is also important to compare query efficiency over data vault and anchor models vs. classical star-like schemas, with a benchmark that allows significant database scale-up.

Finally, various alternative metadata models, independently from the modeling techniques used, could be considered and compared. Schemas of the TECTONIQ documents could also be automatically extracted to enrich the metadata.

Acknowledgments

The authors would like to thank Éric Kergosien, leader of the TECTONIQ project, for making the data corpus available.

References

[1] H. H. Alrehamy and C. Walker. Personal Data Lake With Data Gravity Pull. In *IEEE 5th International Conference on Big Data and Cloud Computing (BDCloud 2015), Dalian, China*, pages 160–167, Washington, DC, USA, 2015. IEEE Computer Society.

[2] C. Bondiombouy and P. Valduriez. Query Processing in Multistore Systems: an overview. Technical Report RR-8890, INRIA Sophia Antipolis-Méditerranée, March 2010.

[3] J. Dixon. Pentaho, Hadoop and Data Lakes. `https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/`, April 2010.

[4] F. A. Eshetu. Data Vault Modelling: An Introductory Guide. B.Sc. Thesis, Helsinki Metropolia University of Applied Sciences, Finland, 2014.

[5] H. Fang. Managing Data Lakes in Big Data Era: What's a data lake and why has it became popular in data management ecosystem. In *5th Annual IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems (CYBER 2015), Shenyang, China*, pages 820–824, June 2015.

[6] P. Ganore. Introduction To The Concept Of Data Lake And Its Benefits. `https://www.esds.co.in/blog/introduction-to-the-concept-of-data-lake-and-its-benefits/`, February 2015.

[7] H. Giménez. PostgreSQL Performance Considerations. `https://robots.thoughtbot.com/postgresql-performance-considerations`, January 2011.

[8] H. Hultgren. Data vault modelling guide – Introductory guide to data vault modelling. Genesee Academy. `https://hanshultgren.files.wordpress.com/2012/09/data-vault-modeling-guide.pdf`, 2012.

[9] B. Inmon. *Data Lake Architecture: Designing the Data Lake and avoiding the garbage dump*. Technics Publications, 2016.

[10] T. John and P. Misra. *Data Lake for Enterprises: Lambda Architecture for building enterprise data systems*. Packt Publishing, May 2017.

[11] V. Jovanovic and I. Bojicic. Conceptual Data Vault Model. In *Southern Association for Information Systems Conference, Atlanta, GA, USA*, pages 131–136. Association for Information Systems, 2012.

[12] E. Kergosien. TEchnologies de l'information et de la communication au Cœur du TerritOire NumérIQue pour la valorisation du patrimoine. `https://tectoniq.meshs.fr/`, 2017.

[13] E. Kergosien, B. Jacquemin, M. Severo, and S. Chaudron. Vers l'interopérabilité des données hétérogènes liées au patrimoine industriel textile. In *18e colloque international sur le document numérique (CIDE18), Montpellier, France*, page 15, 2015.

[14] P. P. Khine and Z. S. Wang. Data Lake: A New Ideology in Big Data Era. In *4th International Conference on Wireless Communication and Sensor Network (WCSN 2017), Wuhan, China*, volume 17 of *ITM Web of Conferences*, pages 1–6, December 2017.

[15] D. Krneta, V. Jovanovic, and Z. Marjanovic. A direct approach to physical Data Vault design. *Computer Science and Information Systems*, 11(2):569–599, 2014.

[16] D. Linstedt. *Super Charge your Data Warehouse: Invaluable Data Modeling Rules to Implement Your Data Vault*. CreateSpace Independent Publishing, 2011.

[17] D. Linstedt. Data Vault Basics. `https://danlinstedt.com/solutions-2/data-vault-basics/`, 2015.

[18] D. Linstedt and M. Olschimke. *Building a Scalable Data Warehouse with Data Vault 2.0*. Morgan Kaufmann, Cambridge, MA, USA, 2015.

[19] N. Miloslavskaya and A. Tolstoy. Big Data, Fast Data and Data Lake Concepts. *Procedia Computer Science*, 88:300–305, 2016.

[20] D. E. O'Leary. Embedding AI and Crowdsourcing in the Big Data Lake. *IEEE Intelligent Systems*, 29(5):70–73, November 2014.

[21] N. Pathirana. Modeling territorial knowledge from web data about natural and cultural heritage. M.Sc. Thesis, Université Lumière Lyon 2, France, 2015.

[22] G. N. Paulley. *Exploiting Functional Dependence in Query Optimization*. PhD thesis, University of Waterloo, Canada, 2000.

[23] O. Regardt, L. Rönnbäck, M. Bergholtz, P. Johannesson, and P. Wohed. Anchor Modeling. In *28th International Conference on Conceptual Modeling (ER 2009), Gramado, Brazil*, volume 5829 of *Lecture Notes in Computer Science*, pages 234–250, Heidelberg, Germany, 2009. Springer.

[24] L. Rönnbäck and H. Hultgren. Comparing Anchor Modeling with Data Vault Modeling. `https://hanshultgren.files.wordpress.com/2013/06/modeling_compare_05_larshans.pdf`, June 2013.

[25] L. Rönnbäck, O. Regardt, M. Bergholtz, P. Johannesson, and P. Wohed. Anchor modeling – Agile information modeling in evolving data environments. *Data and Knowledge Engineering*, 69(12):1229–1253, 2010.

[26] B. Stein and A. Morrison. The enterprise data lake: Better integration and deeper analytics. *Technology Forecast*, 1. http://www.pwc.com/us/en/technology-forecast/2014/cloud-computing/assets/pdf/pwc-technology-forecast-data-lakes.pdf, 2014.

[27] R. Tan, R. Chirkova, V. Gadepally, and T. G. Mattson. Enabling Query Processing across Heterogeneous Data Models: A Survey. In *2017 IEEE International Conference on Big Data (BIGDATA 2017), Boston, USA*, pages 3211–3220, December 2017.

Appendices

A SQL Queries

```
-- 1
SELECT * FROM Hub_Title HT, Sat_Title ST
WHERE HT.Title_id = ST.Title_id
AND Title LIKE '%factory%'

-- 2
SELECT * FROM Link_Document LD, Hub_Title HT, Sat_Title ST,
   Hub_Location HL, Sat_Location SL
WHERE LD.Title_id = HT.Title_id AND LD.Location_id = HL.Location_id
AND HT.Title_id = ST.Title_id AND HL.Location_id = SL.Location_id
AND Title LIKE '%factory%' AND Address = 'Tourcoing'

-- 3
SELECT * FROM Link_Document LD, Hub_Title HT, Sat_Title ST,
   Hub_Location HL, Sat_Location SL, Hub_Date HD, Sat_Date SD
WHERE LD.Title_id = HT.Title_id AND LD.Location_id = HL.Location_id
AND LD.Date_id = HD.Date_id AND HT.Title_id = ST.Title_id
AND HL.Location_id = SL.Location_id AND HD.Date_id = SD.Date_id
AND Title LIKE '%factory%' AND Address = 'Tourcoing'
AND DATE_TRUNC('year', DepositDate) = 2010

-- 4
SELECT * FROM Link_Document LD, Hub_Title HT, Sat_Title ST,
   Hub_Location HL, Sat_Location SL, Hub_Date HD, Sat_Date SD,
```

```
   Hub_Category HC, Sat_Book SB
WHERE LD.Title_id = HT.Title_id AND LD.Location_id = HL.Location_id
AND LD.Date_id = HD.Date_id AND LD.Category_id = HC.Category_id
AND HT.Title_id = ST.Title_id AND HL.Location_id = SL.Location_id
AND HD.Date_id = SD.Date_id AND HC.Category_id = SB.Category_id
AND Title LIKE '%factory%' AND Address = 'Tourcoing'
AND DATE_TRUNC('year', DepositDate) = 2010 AND SB.Category_id = 'book'

-- 5
-- PL/pgSQL is needed here.
DECLARE
   doc RECORD;
   cat RECORD;
BEGIN
   FOR doc IN
      SELECT * FROM Link_Document LD, Hub_Title HT, Sat_Title ST
      WHERE LD.Title_id = HT.Title_id AND HT.Title_id = ST.Title_id
      AND Title LIKE '%factory%
   LOOP
      FOR cat IN EXECUTE
         "SELECT * FROM Hub_Category HC, Sat_" || doc.category_id || " SL
        WHERE HC.Category_id = SL.Category_id"
      LOOP
         -- Tuples doc and cat together provide full document info.
      END LOOP;
   END LOOP;
END;
```

B MongoDB Queries

```
# 1
db.hub_title.aggregate([
        { $match : { 'title': /factory/  }},
        { $lookup : {
                from : "sat_title",
                localField : "id",
                foreignField : "id",
                as : "satellite" }}
]);
```

```
# 2
db.hub_title.aggregate([
        { $match : { 'title': /factory/  }},
        { $lookup : {
                from : "sat_title",
                localField : "id",
                foreignField : "id",
                as : "satellite" }},
{ $lookup : {
                from : "link_document",
                localField : "id",
                foreignField : "id",
                as : "hub_location" }},
        {$unwind : "$link_document"},
        { $lookup : {
                from : "hub_location",
                localField : "id",
                foreignField : "id",
                as : "hub_location" }},
        { $lookup : {
                from : "sat_location",
                localField : "id",
                foreignField : "id",
                as : "sat_location" }},
        {$unwind : "$hub_location"},
        {$match : {'hub_location.location': /Tourcoing/}}
]);

# 3
db.hub_title.aggregate([
        { $match : { 'title': /factory/  }},
        { $lookup : {
                from : "sat_title",
                localField : "id",
                foreignField : "id",
                as : "satellite" }},
{ $lookup : {
                from : "link_document",
```

```
                localField : "id",
                foreignField : "id",
                as : "hub_location" }},
        {$unwind : "$link_document"},
        { $lookup : {
                from : "hub_location",
                localField : "id",
                foreignField : "id",
                as : "hub_location" }},
        { $lookup : {
                from : "sat_location",
                localField : "id",
                foreignField : "id",
                as : "sat_location" }},
        {$unwind : "$hub_location"},
        {$match : {'hub_location.location': /Tourcoing/}},
{$lookup : {
                from : "hub_date",
                localField : "id",
                foreignField : "id",
                as : "hub_date" }},
        {$lookup : {
                from : "sat_date",
                localField : "id",
                foreignField : "id",
                as : "sat_date" }},
        {$unwind : "$hub_date"},
        {$match : {'hub_date.depositdate': /2010/}}
]);

# 4
db.hub_title.aggregate([
      { $match : { 'title': /factory/  }},
      { $lookup : {
               from : "sat_title",
               localField : "id",
               foreignField : "id",
               as : "satellite" }},
      { $lookup : {
```

```
        from : "link_document",
        localField : "id",
        foreignField : "id",
        as : "hub_location" }},
{ $unwind : "$link_document"},
{ $lookup : {
        from : "hub_location",
        localField : "id",
        foreignField : "id",
        as : "hub_location" }},
{ $lookup : {
        from : "sat_location",
        localField : "id",
        foreignField : "id",
        as : "sat_location" }},
{$unwind : "$hub_location"},
{$match : {'hub_location.location': /Tourcoing/}},
{$lookup : {
        from : "hub_date",
        localField : "id",
        foreignField : "id",
        as : "hub_date" }},
{$lookup : {
        from : "sat_date",
        localField : "id",
        foreignField : "id",
        as : "sat_date" }},
{$unwind : "$hub_date"},
{$match : {'hub_date.depositdate': /2010/}},
{$lookup : {
        from : "hub_category",
        localField : "id",
        foreignField : "id",
        as : "hub_category" }},
{$unwind : "$hub_category"},
{$match : {'hub_category.category': /book/}},
{$lookup : {
        from : "sat_books",
        localField : "id",
```

```
                foreignField : "id",
                as : "sat_books" }}
]);

# 5
# Two steps and a function are needed here.
var docs = db.hub_title.aggregate([
      { $match : { 'title': /factory/  }},
      { $lookup : {
                from : "sat_title",
                localField : "id",
                foreignField : "id",
                as : "satellite" }},
      { $lookup : {
                from : "link_document",
                localField : "id",
                foreignField : "id",
                as : "link_document" }},
      {$unwind : "$link_document"},
      {$project : { "cat" : "$link_document.category_id" }
]);
docs.forEach(function(cat){
   db.hub_category.aggregate([
      {$lookup : {
                from : "sat_$cat",
                localField : "id",
                foreignField : "id",
                as : "cat" }},
      {$unwind : "cat"}
   ]);
});
```