Towards a new hybrid approach for building document-oriented data warehouses

Nawfal El Moukhi, Ikram El Azami, Soufiane Hajbi
Department of Computer Science, Faculty of Sciences, Ibn Tofail University, Kenitra, Morocco

Article Info

Article history:

Received Jul 18, 2021
Revised Aug 12, 2022
Accepted Aug 24, 2022

Keywords:

Apache spark
Data warehouse
Document-oriented data base
hybrid approach
Natural language processing
Natural language toolkit
Not only SQL

ABSTRACT

Schemaless databases offer a large storage capacity while guaranteeing high performance in data processing. Unlike relational databases, which are rigid and have shown their limitations in managing large amounts of data. However, the absence of a well-defined schema and structure in not only SQL (NoSQL) databases makes the use of data for decision analysis purposes even more complex and difficult. In this paper, we propose an original approach to build a document-oriented data warehouse from unstructured data. The new approach follows a hybrid paradigm that combines data analysis and user requirements analysis. The first data-driven step exploits the fast and distributed processing of the spark engine to generate a general schema for each collection in the database. The second requirement-driven step consists of analyzing the semantics of the decisional requirements expressed in natural language and mapping them to the schemas of the collections. At the end of the process, a decisional schema is generated in JavaScript object notation (JSON) format and the data loading with the necessary transformations is performed.

Corresponding Author:

Nawfal El Moukhi
Department of Computer Science, Faculty of Sciences, Ibn Tofail University
University Campus, Kenitra, P.O. Box 133, Morocco
Email: elmoukhi.nawfal@gmail.com

1. INTRODUCTION

Since the 1980s, relational database management systems (DBMSs) have continued to grow in importance compared to other data management systems. Today, still used by the majority of companies, they are appreciated for their ability to ensure strong data consistency and guarantee reliability during transactions through the atomic-consistent-isolated-durable (ACID) rules adopted by these systems [1]. However, the advent of the big data era and the explosion of data volumes to be processed have led companies to rethink their data management models, especially as relational DBMSs have shown many limitations in terms of distributed architecture. As a result, large companies such as Google, Amazon and Facebook have embarked on several research projects that have given rise to several not only SQL (NoSQL) database management systems, focusing on performance, reliability and consistency. These new systems are characterized by the absence of a priori defined schema, which gives them great flexibility, and a distributed architecture allowing horizontal scalability and a huge gain in processing time.

Today, NoSQL DBMSs are gaining in popularity [2], and there is no doubt that they will one day be the most widely used systems in the world. Therefore, it has become necessary to rethink the design of data warehouses to accommodate the new NoSQL model. The first research works [3]–[7] addressing this issue were limited to developing methods to transform a relational data warehouse into a NoSQL data warehouse by using model-driven architecture (MDA) rules and techniques. Although these methods are very important for upgrading business intelligence systems to the new NoSQL logic, the issue of designing a data warehouse from unstructured data sources still remains problematic. In this sense, researchers [8], [9] suggested to transform the NoSQL database into a relational database. The latter can be used to generate a multidimensional data warehouse using classical methods. Even if these methods have succeeded in integrating unstructured data sources, they do not allow to exploit the performance of NoSQL systems and may have many limitations when dealing with large amounts of data. Other research works [10]–[13] have used the MapReduce framework proposed by Google in 2004 [14] to generate a NoSQL data warehouse. These methods represent a great progress in the field, since they allow to solve the problem while remaining in the NoSQL logic throughout the design process. However, the MapReduce framework has several limitations [15] and the paradigm (data-driven) adopted by these methods neglects the users' requirements, hence the interest of the present work.

In this paper, we introduce a new hybrid method that allows to generate a NoSQL data warehouse by considering both user requirements and data sources thus adopting the best paradigm to design a data warehouse [16]. The new approach uses a document-oriented database as a data source and allows first to analyze the different JSON files in order to detect the structure of each collection and to extract its schema. The second phase consists in mapping decisional needs expressed in natural language to the structures of the collections using natural language processing techniques. At the end of the process, a multidimensional model is generated and integrated into a NoSQL data warehouse for each expressed need. The paper is organized: in section 2, we present the main steps of the new method. In section 3, we use a real database to test the new method and discuss the results. Section 4 is devoted to draw conclusions and present research perspectives.

2. RESEARCH METHOD

In this paper, we propose a new hybrid method for generating a data warehouse that can meet the decision-making needs of decision-makers. In the first phase, the new method exploits the speed of the spark framework-which is much faster than MapReduce according to several research works [17]–[26] in analyzing large amounts of unstructured and distributed data, to generate a general schema for each collection. This allows to extract the structure of a large amount of data in a reasonable time, thus revealing the richness of the data stored in a document-oriented database. In the second phase, decisional needs expressed in natural language by the decision-makers are semantically analyzed and compared with the different general schemas extracted by the first data-driven phase. Such a mapping allows to identify the collection that best meets a decisional need, and to select only those fields that are useful for the user. Figure 1 shows the main steps of the new hybrid approach.

As shown in the figure, the new hybrid approach consists of 4 main steps:

a. Schemas extraction: a document-oriented database is composed of several collections, and each collection contains JavaScript object notation (JSON) (or XML) documents distributed across multiple partitions. This first step consists in exploring and reading the JSON files of a collection, contained in each partition. At the end of the step, the schema (*DataFrame*) of each partition of the collection is extracted, thus obtaining a multitude of schemas for the same collection.
b. Merging schemas: since we are working on a NoSQL database and the source data is unstructured, the schemas of the various partitions are different and have common fields and new fields. Therefore, it is essential to merge the schemas to get a first version of the general schema of the collection. Thus, the common fields must be retained once and the new fields added. A simple merging of the different extracted schemas will generate errors for the following reasons: i) some fields do not have the same data types, ii) there is a significant change in the structure of some fields, and iii) the *union()* method can only merge schemas with the same structure. To successfully merge the schemas, we first need to convert the columns to strings to make sure the data types are compatible. Since JSON RDD allows the union of schemas even when the structures are different, we need to convert the *DataFrame* to JSON RDD using the *toJSON()* method, before uniting the partition schemas. In this way, we avoid the second and third errors.
c. Merging fields: The first version of the general schema is the result of merging the schemas of the partitions. As a result, identical fields with different names are also retained. This step remedies this problem by analyzing the semantics of the names, using the NLTK library of python, and comparing them for similarities. The process of merging fields is done in stages and by tree level of the general schema. Thus, the fields at the highest level of the tree structure are analyzed and compared first. Fields with a score of 50% or more are proposed to the designer, and only after having made the necessary mergers that the program move on to the second level and so on, until the last level of the tree. Figure 2 shows the process of merging fields.

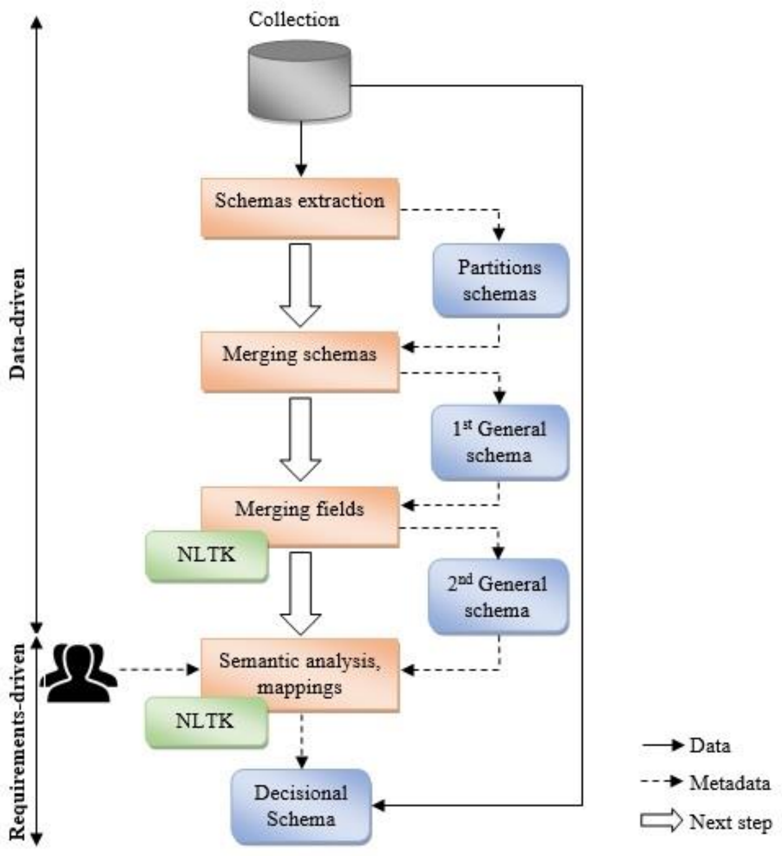


Figure 1. The new hybrid approach process

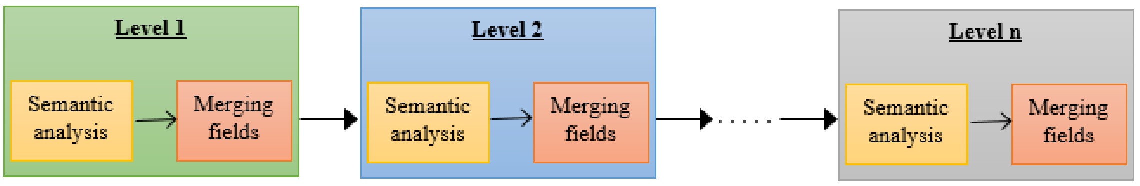


Figure 2. The new hybrid approach process

d. Semantic analysis and mappings: In this step, a decisional need expressed in natural language is compared semantically to the general schemas of the collections. In order to perform this comparison, a collection sentence is generated from each general schema by concatenating the collection name and the field names. Thus, the decisional need and the collections sentences are semantically analyzed and compared and each comparison results in a similarity score. Only collections sentences with a score higher than or equal to 50% are retained and the corresponding general schemas are proposed to the user. The general schema proposed by the system and selected by the user will enable him, on the one hand, to check the availability of the data he needs for decision making, and on the other hand, to discover other fields likely to interest him and to meet a latent and unexpressed decision-making need. After selecting the relevant fields, a decisional schema that meets the user's need is generated in JSON format and this schema is used to load the data. When loading the data, the necessary operations induced by the step of merging fields, must be applied. Thus, for the merged fields, the data of the non-retained fields must be loaded into the retained ones. At the end of the process, we obtain a version of the collection intended specifically for decision-making use and meeting a specific decisional need. All the collections that meet the decisional needs expressed by the users will form our NoSQL data warehouse. Figure 3(a) shows the algorithm of the first data-driven phase and Figure 3(b) the algorithm of the second requirements-driven phase.

```
Algorithm 1. The first data-driven phase
FOR each collection
    FOR each partition
        extract the Dataframe schema of the json files
        convert the columns to strings
        convert the dataframe to json rdd
    END FOR
    generate the 1st version of the general schema by merging the schemas of the partitions
    FOR (level=1; level<=max  level; level++)
        FOR each field
                tokenize the name
                    FOR each token
                        extract the synsets
                    END FOR
            END FOR
     calculate the similarity score between each two fields name using the extracted synsets
     RETURN fields with a similarity score>=50%
     generate the 2nd version of the general schema by merging similar fields
     save the 2nd version of the general schema in json format
     END FOR
END FOR
```

(a)

```
Algorithm 2. The second requirements-driven phase
FOR each general schema of a collection
    extract the fields names
    generate the collection sentence by concatenating the collection name and the field names
    RETURN the collection sentence
END FOR
FOR each decisional need expressed in natural language
    tokenize the decisional need sentence
    FOR each token
        extract the synsets
    END FOR
    FOR each collection sentence
        tokenize the collection sentence
        FOR each token
            extract the synsets
        END FOR
    END FOR
    calculate the similarity score between the decisional need sentence and each collection sentence using the extracted synsets
    RETURN collections with a similarity score>= 50%
END FOR
```

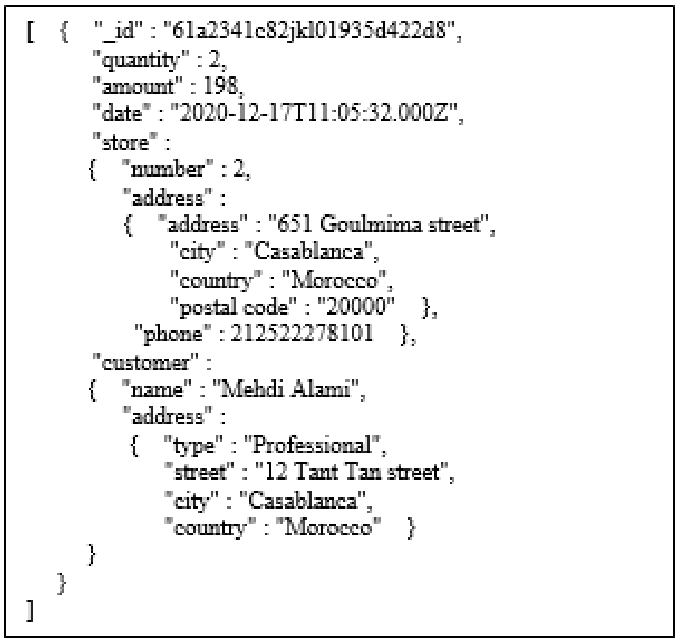
(b)

Figure 3. Algorithm of the new hybrid approach (a) algorithm 1: the first data-driven phase, and (b) algorithm 2: the second requirements-driven phase

3. RESULTS AND DISCUSSION

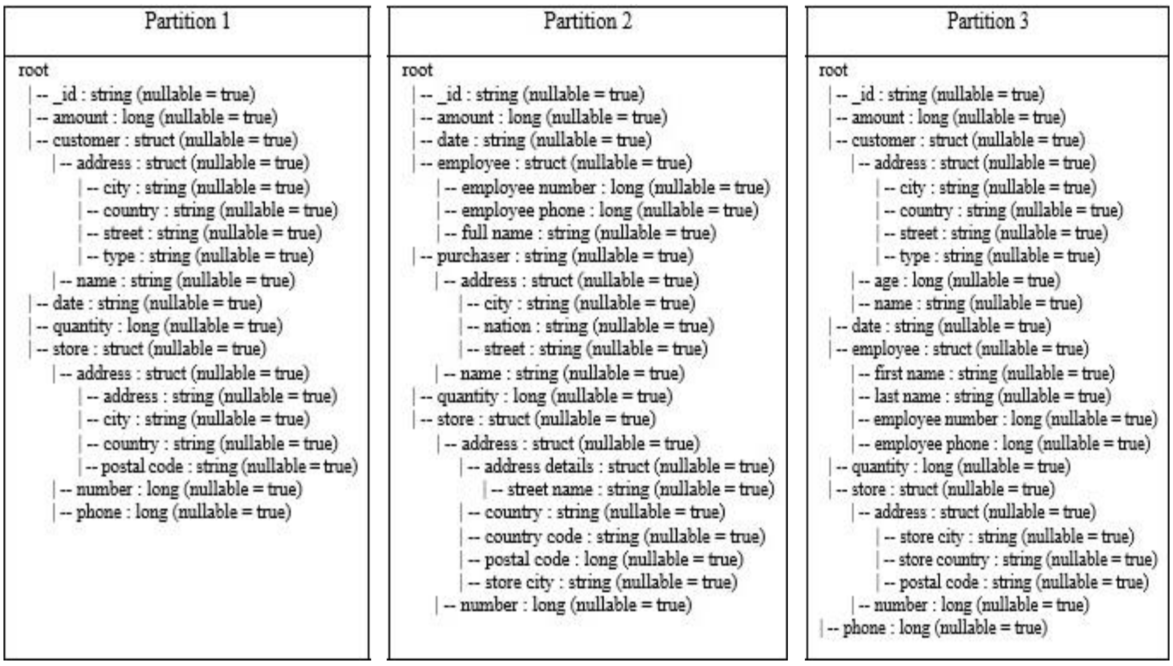
In order to illustrate each step of the new approach, we used a real-world example of a sales collection containing various data and distributed across three partitions. Obviously, the document is the central concept of a document-oriented database, which encapsulates and encodes its data in some standard format. The most widely adopted format is JSON, which we use as a reference in this work. Figure 4 shows a document from the "Sales" collection contained in partition 1. The document is organized according to two nesting levels: i) the first level contains information about a sale made at a given time. In addition to the "date" and time, it contains data about the "amount" paid by the customer and the "quantity" purchased and ii) the second level contains two main fields: "store" containing information about the store that made the sale, and "customer" containing information about the customer who bought from the store.

The extraction of the collection schema first involves the extraction of the schema of each partition of the collection. Figure 5 shows the extracted schemas of the "Sales" collection with the schema of partition 1 in Figure 5(a), the schema of partition 2 in Figure 5(b) and the schema of partition 3 in Figure 5(c). After merging the schemas, we will get the schema in Figure 6 containing all the fields with the *dataframe* format in Figure 6(a), and the JSON format in Figure 6(b). Table 1 presents the similar fields proposed by the system for a possible merger.



```
[  {   "_id" : "61a2341c82jkl01935d422d8",
       "quantity" : 2,
       "amount" : 198,
       "date" : "2020-12-17T11:05:32.000Z",
       "store" :
       {   "number" : 2,
           "address" :
           {   "address" : "651 Goulmima street",
                   "city" : "Casablanca",
                   "country" : "Morocco",
                   "postal code" : "20000"   },
             "phone" : 212522278101   },
       "customer" :
       {   "name" : "Mehdi Alami",
           "address" :
            {   "type" : "Professional",
                "street" : "12 Tant Tan street",
                "city" : "Casablanca",
                "country" : "Morocco"   }
       }
   }
]
```

Figure 4. Example of a document from the “Sales” collection



Partition 1

```
root
 |-- _id : string (nullable = true)
 |-- amount : long (nullable = true)
 |-- customer : struct (nullable = true)
     |-- address : struct (nullable = true)
         |-- city : string (nullable = true)
         |-- country : string (nullable = true)
         |-- street : string (nullable = true)
         |-- type : string (nullable = true)
     |-- name : string (nullable = true)
 |-- date : string (nullable = true)
 |-- quantity : long (nullable = true)
 |-- store : struct (nullable = true)
     |-- address : struct (nullable = true)
         |-- address : string (nullable = true)
         |-- city : string (nullable = true)
         |-- country : string (nullable = true)
         |-- postal code : string (nullable = true)
     |-- number : long (nullable = true)
     |-- phone : long (nullable = true)
```

(a)

Partition 2

```
root
 |-- _id : string (nullable = true)
 |-- amount : long (nullable = true)
 |-- date : string (nullable = true)
 |-- employee : struct (nullable = true)
     |-- employee number : long (nullable = true)
     |-- employee phone : long (nullable = true)
     |-- full name : string (nullable = true)
 |-- purchaser : string (nullable = true)
     |-- address : struct (nullable = true)
         |-- city : string (nullable = true)
         |-- nation : string (nullable = true)
         |-- street : string (nullable = true)
     |-- name : string (nullable = true)
 |-- quantity : long (nullable = true)
 |-- store : struct (nullable = true)
     |-- address : struct (nullable = true)
         |-- address details : struct (nullable = true)
             |-- street name : string (nullable = true)
         |-- country : string (nullable = true)
         |-- country code : string (nullable = true)
         |-- postal code : long (nullable = true)
         |-- store city : string (nullable = true)
     |-- number : long (nullable = true)
```

(b)

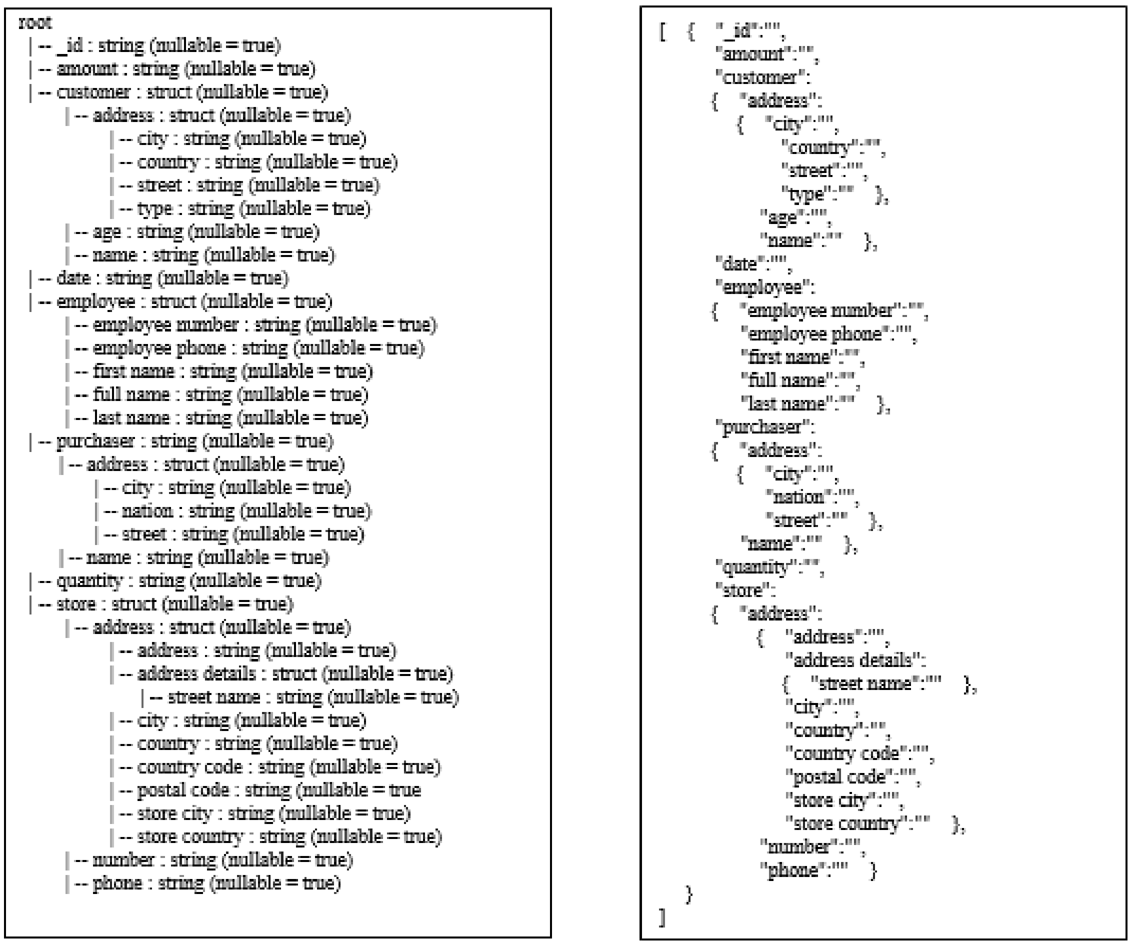
Partition 3

```
root
 |-- _id : string (nullable = true)
 |-- amount : long (nullable = true)
 |-- customer : struct (nullable = true)
     |-- address : struct (nullable = true)
         |-- city : string (nullable = true)
         |-- country : string (nullable = true)
         |-- street : string (nullable = true)
         |-- type : string (nullable = true)
     |-- age : long (nullable = true)
     |-- name : string (nullable = true)
 |-- date : string (nullable = true)
 |-- employee : struct (nullable = true)
     |-- first name : string (nullable = true)
     |-- last name : string (nullable = true)
     |-- employee number : long (nullable = true)
     |-- employee phone : long (nullable = true)
 |-- quantity : long (nullable = true)
 |-- store : struct (nullable = true)
     |-- address : struct (nullable = true)
         |-- store city : string (nullable = true)
         |-- store country : string (nullable = true)
         |-- postal code : string (nullable = true)
     |-- number : long (nullable = true)
|-- phone : long (nullable = true)
```

(c)

Figure 5. The schemas of partitions of the “Sales” collection (a) schema of partition 1, (b) schema of partition 2, and (c) schema of partition 3

In total, the system proposes to the user to merge 26 fields, and 16 fields are to be merged (61.5% of the total proposed). The fields “employee number”/”employee phone”, “country”/”country code”, “country code”/”Postal code”, “country code”/”store country”, “store city”/”store country” refer to different information despite the similarity calculated. For the 2 fields “first name” and “last name”, they must be merged into a single “full name” field. When loading the data, a concatenation of the data of the fields “first name” and “last name” should be considered. Figure 7 shows the second general schema of the collection in JSON format, after performing the necessary mergers.



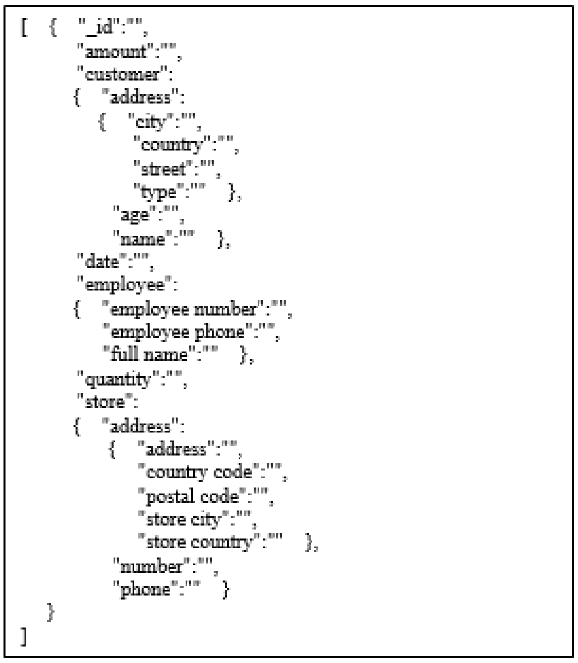
```
root
 | -- _id : string (nullable = true)
 | -- amount : string (nullable = true)
 | -- customer : struct (nullable = true)
      | -- address : struct (nullable = true)
            | -- city : string (nullable = true)
            | -- country : string (nullable = true)
            | -- street : string (nullable = true)
            | -- type : string (nullable = true)
      | -- age : string (nullable = true)
      | -- name : string (nullable = true)
 | -- date : string (nullable = true)
 | -- employee : struct (nullable = true)
      | -- employee number : string (nullable = true)
      | -- employee phone : string (nullable = true)
      | -- first name : string (nullable = true)
      | -- full name : string (nullable = true)
      | -- last name : string (nullable = true)
 | -- purchaser : string (nullable = true)
     | -- address : struct (nullable = true)
          | -- city : string (nullable = true)
          | -- nation : string (nullable = true)
          | -- street : string (nullable = true)
     | -- name : string (nullable = true)
 | -- quantity : string (nullable = true)
 | -- store : struct (nullable = true)
      | -- address : struct (nullable = true)
            | -- address : string (nullable = true)
            | -- address details : struct (nullable = true)
                 | -- street name : string (nullable = true)
            | -- city : string (nullable = true)
            | -- country : string (nullable = true)
            | -- country code : string (nullable = true)
            | -- postal code : string (nullable = true
            | -- store city : string (nullable = true)
            | -- store country : string (nullable = true)
      | -- number : string (nullable = true)
      | -- phone : string (nullable = true)
```

(a)

```
[  {   "_id":"",
       "amount":"",
       "customer":
      {   "address":
          {   "city":"",
                "country":"",
                "street":"",
                "type":""    },
            "age":"",
            "name":""    },
       "date":"",
       "employee":
      {   "employee number":"",
          "employee phone":"",
          "first name":"",
          "full name":"",
          "last name":""    },
       "purchaser":
      {   "address":
          {   "city":"",
              "nation":"",
              "street":""    },
          "name":""    },
       "quantity":"",
       "store":
      {   "address":
            {   "address":"",
                "address details":
               {   "street name":""    },
                "city":"",
                "country":"",
                "country code":"",
                "postal code":"",
                "store city":"",
                "store country":""    },
            "number":"",
            "phone":""    }
   }
]
```

(b)

Figure 6. The 1st general schema in (a) *dataframe* format and (b) JSON format



```
[  {   "_id":"",
       "amount":"",
       "customer":
      {   "address":
          {   "city":"",
                "country":"",
                "street":"",
                "type":""    },
            "age":"",
            "name":""    },
       "date":"",
       "employee":
      {   "employee number":"",
          "employee phone":"",
          "full name":""    },
       "quantity":"",
       "store":
      {   "address":
            {   "address":"",
                "country code":"",
                "postal code":"",
                "store city":"",
                "store country":""    },
            "number":"",
            "phone":""    }
   }
]
```

Figure 7. 2nd general schema in JSON format

Table 1. The fields proposed for merging after the semantic comparison operation

Field 1	Field 2
purchaser	customer
country	nation
employee number	employee phone
first name	full name
first name	last name
full name	last name
address	address details
city	store city
country	country code
country	store country
country code	postal code
country code	store country
store city	store country

We assume that the user has expressed the following decisional need: "*We want to analyze the quantity sold by date, by store, by city and by country*". The combination of our "sales" collection gives the following result: "*sales, amount, date, quantity, customer, address, city, country, street, type, age, name, employee, employee number, employee phone, full name, store, address, country code, postal code, store city, store country, number, phone*". We also assume that we have two other collections with the following combinations: i) collection "suppliers": suppliers, number, name, phone, address, city, country, cost, quality, date, delivery number, mode, total demand; ii) collection "Products": products, number, name, Reference, Description, color, type, width, height, weight, production material, popularity, quality, price. Table 2 summarizes the results of the semantic comparisons between the expressed need and the 3 collections. At the end of the process, a decision schema that meets the user's need is generated. Figure 8 shows this decision schema in JSON format.

Table 2. scores of the semantic comparison between the need expressed by the user and the collections

	Sales	Suppliers	Products
"We want to analyze the quantity sold by date, by store, by city and by country"	0.733	0.527	0.147

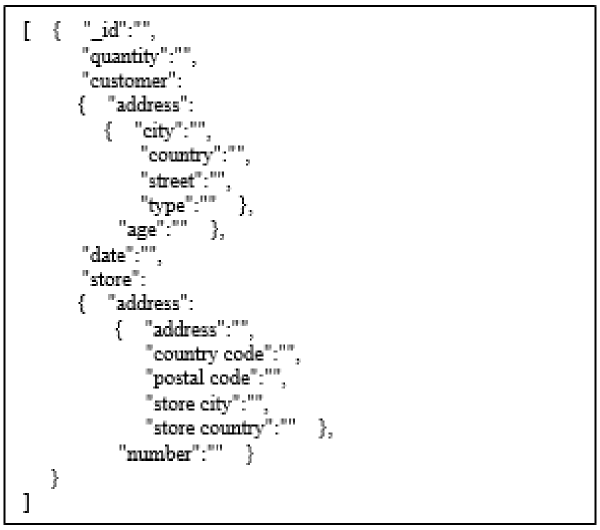

```
[  {  "_id":"",
      "quantity":"",
      "customer":
      {  "address":
         {  "city":"",
            "country":"",
            "street":"",
            "type":""  },
         "age":""  },
      "date":"",
      "store":
      {  "address":
         {  "address":"",
            "country code":"",
            "postal code":"",
            "store city":"",
            "store country":""  },
         "number":""  }
   }
]
```

Figure 8. Decisional schema in JSON format

4. CONCLUSION

In this paper, we presented a new hybrid approach to generate a document-oriented data warehouse from unstructured data sources. The new approach is not limited to the analysis of data sources but also includes the analysis of needs, hence its hybrid nature. The semantic analysis of the decisional need and its comparison with the general schemas allows the system to assist the user in the choice of the collection and the fields to be retained in the final decisional schema. In this way, only data useful for decision-making will be loaded into the data warehouse and useless data will be eliminated. The prior knowledge of the structure of the decision schema and the tree structure of the fields greatly facilitates the formulation of decision queries in the document-oriented data warehouse. Our research perspectives consist in integrating other types of NoSQL data sources such as column-oriented databases or key/value-oriented databases.

REFERENCES

[1] S. Shetty, B. D. Rao, and S. Prabhu, "Growth of relational model: interdependence and complementary to big data," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 11, no. 2, pp. 1780–1795, Apr. 2021, doi: 10.11591/ijece.v11i2.pp1780-1795.

[2] "DBMS popularity broken down by database model Number of systems per category, December 2020." 2021.

[3] F. Abdelhedi, A. Ait Brahim, F. Atigui, and G. Zurfluh, "MDA-based approach for NoSQL databases modelling," in *Big Data Analytics and Knowledge Discovery*, Springer International Publishing, 2017, pp. 88–102., doi: 10.1007/978-3-319-64283-3_7.

[4] M. Chevalier, M. El Malki, A. Kopliku, O. Teste, and R. Tournier, "Implementation of multidimensional databases with document-oriented NoSQL," in *Big Data Analytics and Knowledge Discovery*, Springer International Publishing, 2015, pp. 379–390., doi: 10.1007/978-3-319-22729-0_29.

[5] M. Y. Santos, B. Martinho, and C. Costa, "Modelling and implementing big data warehouses for decision support," *Journal of Management Analytics*, vol. 4, no. 2, pp. 111–129, Apr. 2017, doi: 10.1080/23270012.2017.1304292.

[6] F. Abdelhedi, A. A. Brahim, and G. Zurfluh, "Formalizing the mapping of UML conceptual schemas to column-oriented databases," *International Journal of Data Warehousing and Mining*, vol. 14, no. 3, pp. 44–68, Jul. 2018, doi: 10.4018/IJDWM.2018070103.

[7] K. Dehdouh, F. Bentayeb, O. Boussaid, and N. Kabachi, "Using the column oriented NoSQL model for implementing big data warehouses," 2015.

[8] Z. Aftab, W. Iqbal, K. M. Almustafa, F. Bukhari, and M. Abdullah, "Automatic NoSQL to relational database transformation with dynamic schema mapping," *Scientific Programming*, vol. 2020, pp. 1–13, Jul. 2020, doi: 10.1155/2020/8813350.

[9] B. Maity, A. Acharya, T. Goto, and S. Sen, "A framework to convert NoSQL to relational model," in *Proceedings of the 6th ACM/ACIS International Conference on Applied Computing and Information Technology-ACIT 2018*, 2018, pp. 1–6., doi: 10.1145/3265007.3265011.

[10] S. Bouaziz, A. Nabli, and F. Gargouri, "Design a data warehouse schema from document-oriented database," *Procedia Computer Science*, vol. 159, pp. 221–230, 2019, doi: 10.1016/j.procs.2019.09.177.

[11] K. Ma and B. Yang, "Introducing extreme data storage middleware of schema-free document stores using MapReduce," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 20, no. 4, 2015, doi: 10.1504/IJAHUC.2015.073439.

[12] H. Chai, G. Wu, and Y. Zhao, "A document-based data warehousing approach for large scale data mining," in *Pervasive Computing and the Networked World*, Springer Berlin Heidelberg, 2013, pp. 69–81., doi: 10.1007/978-3-642-37015-1_7.

[13] K. Dehdouh, O. Boussaid, and F. Bentayeb, "Big data warehouse," *International Journal of Decision Support System Technology*, vol. 12, no. 1, pp. 1–24, Jan. 2020, doi: 10.4018/IJDSST.2020010101.

[14] M. Dayalan, "MapReduce: simplified data processing on large cluster," *International Journal of Research and Engineering*, vol. 5, no. 5, pp. 399–403, Apr. 2018, doi: 10.21276/ijre.2018.5.5.4.

[15] S. A. Thanekar, K. Subrahmanyam, and A. B. Bagwan, "Big data and MapReduce challenges, opportunities and trends," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 6, no. 6, pp. 2911–2919, Dec. 2016, doi: 10.11591/ijece.v6i6.10555.

[16] N. El Moukhi, I. El, A. Mouloudi, and A. Elmounadi, "Merge of X-ETL and XCube towards a standard hybrid method for designing data warehouses," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 10, 2019, doi: 10.14569/IJACSA.2019.0101019.

[17] Z. Ruihong and H. Zhihua, "WITHDRAWN: comparative research on active learning of big aata based on mapreduce and spark," *Microprocessors and Microsystems*, Nov. 2020, doi: 10.1016/j.micpro.2020.103425.

[18] X. Wu and S. Ji, "Comparative study on MapReduce and spark for big data analytics," *Journal of Software*, vol. 29, no. 6, pp. 1770–1791, 2018

[19] E. P. S. Castro, T. D. Maia, M. R. Pereira, A. A. A. Esmin, and D. A. Pereira, "Review and comparison of Apriori algorithm implementations on Hadoop-MapReduce and Spark," *The Knowledge Engineering Review*, vol. 33, Jul. 2018, doi: 10.1017/S0269888918000127.

[20] J. Shi *et al.*, "Clash of the titans," *Proceedings of the VLDB Endowment*, vol. 8, no. 13, pp. 2110–2121, Sep. 2015, doi: 10.14778/2831360.2831365.

[21] N. Ahmed, A. L. C. Barczak, T. Susnjak, and M. A. Rashid, "A comprehensive performance analysis of apache hadoop and apache spark for large scale data sets using HiBench," *Journal of Big Data*, vol. 7, no. 1, p. 110, Dec. 2020, doi: 10.1186/s40537-020-00388-5.

[22] I. Mavridis and H. Karatza, "Performance evaluation of cloud-based log file analysis with apache hadoop and apache spark," *Journal of Systems and Software*, vol. 125, pp. 133–151, Mar. 2017, doi: 10.1016/j.jss.2016.11.037.

[23] A. Mostafaeipour, A. Jahangard Rafsanjani, M. Ahmadi, and J. Arockia Dhanraj, "Investigating the performance of Hadoop and Spark platforms on machine learning algorithms," *The Journal of Supercomputing*, vol. 77, no. 2, pp. 1273–1300, Feb. 2021, doi: 10.1007/s11227-020-03328-5.

[24] J. Veiga, R. R. Exposito, X. C. Pardo, G. L. Taboada, and J. Tourifio, "Performance evaluation of big data frameworks for large-scale data analytics," in *2016 IEEE International Conference on Big Data (Big Data)*, Dec. 2016, pp. 424–431., doi: 10.1109/BigData.2016.7840633.

[25] Y. Benlachimi, A. El, and M. Lahcen, "A comparative analysis of hadoop and spark frameworks using word count algorithm," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 4, 2021, doi: 10.14569/IJACSA.2021.0120495.

[26] E. L. Lydia, R. M. Vidhyavathi, I. Pustokhina, and D. A. Pustokhin, "Comparative performance analysis of apache spark and map reduce using k-means," *International Journal on Emerging Technologies*, vol. 11, no. 2, pp. 198–204, 2020,

BIOGRAPHIES OF AUTHORS

Nawfal El Moukhi was born in Salé in 1987. He graduated from the School of Information Science of Rabat and he got a Master's degree in applied computer science from Mohammed V University in 2013. He holds a PhD in computer science from Ibn Tofail University in Kenitra, Morocco. His research interests are data warehousing, data mining and big data. He can be contacted at email: elmoukhi.nawfal@mail.com.

Ikram El Azami received his PhD degree from the University of Valenciennes and Hainaux Combrésis, France, and The University of Sidi Mohammed Ben Abdellah, Fez, Morocco. He is currently full professor in the Department of Computer Science at the Faculty of Sciences, Ibn Tofail University of Kenitra. He's a member of the MIS (Multimedia and Information Systems) research Team of MISC Laboratory (Communication Systems and Information Modeling), Faculty of Sciences of Ibn Tofail University. His research interests are information systems engineering, business intelligence, big data and cloud computing. He can be contacted at email: akram_elazami@yahoo.fr.

Soufiane Hajbi is a computer engineer. Currently, he is a PhD student at Ibn Tofail University in Kenitra, Morocco. His research interests are natural language processing, data warehousing, data mining and big data. He can be contacted at email: soufiane.hajbi@uit.ac.ma.