OPEN ACCESS
ISSN 2460-9056
http://socj.telkomuniversity.ac.id/ijoict/

Intl. Journal on ICT
Vol. 7, No. 2, Dec 2021. pp. 36-47
doi: 10.21108/ijoict.v7i2.584

A Schema Extraction of Document-Oriented Database for Data Warehouse

A. Nurul Istiqamah[1], Kemas Rahmat Saleh Wiharja[2*]

[12]*School of Computing, Telkom University*
Jl. Telekomunikasi 1 Terusan Buah Batu, Bandung, Jawa Barat, Indonesia

[*] bagindokemas@telkomuniversity.ac.id

Abstract

The data warehouse is a very famous solution for analyzing business data from heterogeneous sources. Unfortunately, a data warehouse only can analyze structured data. Whereas, nowadays, thanks to the popularity of social media and the ease of creating data on the web, we are experiencing a flood of unstructured data. Therefore, we need an approach that can "structure" the unstructured data into structured data that can be processed by the data warehouse. To do this, we propose a schema extraction approach using Google Cloud Platform that will create a schema from unstructured data. Based on our experiment, our approach successfully produces a schema from unstructured data. To the best of our knowledge, we are the first in using Google Cloud Platform for extracting a schema. We also prove that our approach helps the database developer to understand the unstructured data better.

Keywords: data warehouse, schema extraction, unstructured data

I. INTRODUCTION

IN the last few years, we have seen a dramatic increase in the volume of unstructured data. The development of web 2.0 and 3.0 and the increasing use of the internet by many individuals and organizations have resulted in the size of unstructured data having exceeded dramatically that of structured data. Most business organizations store unstructured data in document format. These documents are essential in the decision-making process [1].

Unstructured data is represented by "schema less" data, meaning that each instance in a collection has a different "local" schema. An instance can have a different structure even though it can have the same concept or specific features from other instance attributes. Thus, schema less databases are preferred for storing heterogeneous data with variable schemas and structural forms [2].

For analyzing the data and producing valuable insights, a company or an organization utilizes a data warehouse. However, since the data warehouse needs to operate on structured data, we need an approach that can transform the unstructured data into structured data so it can be stored, analyzed, and managed by a data warehouse.

[3] extracts schema from unstructured data using Hadoop Distributed File System (HDFS). We notice from their work, that HDFS has the following limitations: (1). HDFS running on a server, not on the cloud, (2). one needs to estimate in advance the requirement of storage and processor prior running the HDFS, (3). One needs to do the estimation precisely, otherwise it will be a wasted budget, (4) Map Reduce jobs in HDFS cannot be started until we started the Name Node exit safe mode; (5) Hadoop requires regular maintenance; (6) Hadoop requires a pretty complex configuration to use other services; (7) Data is stored in computer/server node clusters so that they are vulnerable to a single-point failure.

Received on 06 Oct 2021, Revised on 31 Oct 2021, Accepted and Published on 31 Dec 2021

After we understood that HDFS is not the best choice, we decided to explore similar platforms such as Google Cloud Platform (GCP). We found out that GCP is better than HDFS for identifying the graphic structure of an unstructured document and generating a schema for the following reasons: (1) GCP costs less because users only pay for what they use rather than over-buying hardware; (2) GCP can start the task node as soon as the task node starts; (3) in GCP, users do not need to perform routine maintenance because the site reliability engineering (SRE) team from Google has done this work for users; (4) GCP has better interoperability as users can easily manage data stored in Big Query using other GCP services such as Dataflow; (5) GCP has higher data availability because GCP cloud storage is not prone to single point failure or even cluster failure.

We also found out that hitherto, there is no research that uses GCP for generating schema. Therefore, our contributions are twofold: (1). We are the first in using GCP for extracting schema, and (2). We prove that the usage of schemas can help developers understand the unstructured data better and faster.

In this research, we use Big Query to process data and extract schemas. As for the datasets, we use bibliographic information from Database System and Logic Programming (DBLP)[1] and Microsoft Academic Knowledge Graph (MAKG)[2].

The rest of the paper is structured as follows. Section 2 presents some related work related to our topic, section 3 explains our approach, section 4 describes the results obtained in this study, and section 5 shows the conclusion from this research.

II. Related Works

In this section, we review several important works that are related to our paper. We begin our discussion with explaining NoSQL, Data Warehouse, and then Schema Extraction from Unstructured Data.

A. NoSQL

According to [4], "NoSQL is a non-relational database management system, faster retrieval of information from databases." One can relate NoSQL with its predecessor, the relational database system. NoSQL databases' characteristics are often used in non-relational, open-source distributed, and high-performance databases linearly. Therefore, NoSQL does not organize data in relational tables, and everyone can access the source code freely, update it as needed and compile it.

There are four data models in NoSQL data models, such as document-oriented, column-oriented, object-oriented, and graph-oriented [5].

1) Document-Oriented Database: According to Oxford Dictionary [6], a document is "a piece of written, printed, or electronic matter that provides information or evidence or that serves as an official record." The examples of documents are as follows: a book, a form, a card, or a meeting summary. Simply put, a document-oriented database is a database that we use to store and organize documents. In this type of database, the data hierarchy is the pair of keys and values, document, and collection (group of a similar document. Whereas in relational databases, the data hierarchy is row, table, and database (relation among tables).

2) *Column-Oriented Database:* A column-oriented or columnar database is a database that stores data tables by column rather than row. In simple words, a column-oriented database is a transpose of a row-oriented database [7].

For example, we give a simple illustration in Table I. This table includes an employee identifier (EmpId) and name fields (LastName and FirstName).

[1] https://dblp.org/
[2] https://makg.org/

TABLE I
A SIMPLE ILLUSTRATION OF EMPLOYEE TABLE

RowId	EmpId	FirstName	LastName
001	10	John	Doe
002	12	Jane	Doe
003	22	Richard	Roe

A row-oriented database system is designed to return data for an entire row or record efficiently. A row-oriented database matches the typical use case where the system attempts to retrieve information about a particular object. A standard method of storing a table is to serialize each row of data, like Fig. 1.

```
001:10,John,Doe;
002:12,Jane,Doe;
003:22,Richard,Roe;
```

Fig. 1 How a row-oriented database stores data

In a column-oriented database, all values in a column are physically grouped. For example, all values in column 1 are grouped, then all column 2 are grouped, etc. It allows specific data elements, like FirstName, to be accessed in columns as a group rather than individually row-by-row. For our example table, a column-oriented database would store the data like Fig. 2.

```
10:001,12:002,22:003;
John:001,Jane:002,Richard:003;
Doe:001,Doe:002,Roe:003;
```

Fig. 2 How a column-oriented database stores data

3) Graph-Oriented Database: A graph database (GraphDB) is a database that uses a graph structure containing nodes, edges, and properties to represent and store information. GraphDB is needed for large-scale graph data, especially network biology researchers and social networking sites like Facebook and Twitter. GraphDB maps objects directly to applications and is more intuitive to describe associative data sets [4].

B. Data Warehouse

A data warehouse is a relational database designed for query and analysis purposes. The data warehouse contains historical data originating from transaction data sources and various other data sources. Data warehouses separate analytical workloads from transactions and allow companies to combine data from multiple sources [8].

Data warehouse supports the provision of appropriate data and transforms data into highly accurate information and can be used to find patterns and facts to identify problems and provide data and information at the right time [9].

To be able to analyze the data in the data warehouse, a schema is needed. The structured data schema is in the ER Diagram, which is not discussed in this study. Meanwhile, the schema of the unstructured data is explained in the next sub-chapter.

C. Schema Extraction from Unstructured Data

A schema can be equivalent to metadata because it has essential information in a piece of data such as structure, timing relationships, etc. In an unstructured data, the schema is in the data or document itself, we call it a local schema. To find out the arrangement of data or schemas in a document-oriented database, we need an approach called schema extraction.

In this section, we review the existing work on schema extraction from a NoSQL schemeless database. [10] proposes an extraction, transformation, and loading strategy to integrate unstructured data into a data

warehouse—an approach based on the MapReduce programming system for parallelizing ETL streams and the Hadoop paradigm. ETL tools are frequently used in data warehousing, where data is assembled from different sources to perform data analytics. The unstructured data is extracted using the MapReduce program and stored in the HDFS, where it is processed, analyzed, and transformed by applying specific filters. Finally, the data is loaded into the organization's data warehouse. However, this approach does not show the heterogeneity of the dataset.

Similarly with [10], [11] developed the ETL process for unstructured data in a data warehouse using the MapReduce programming system. They use Pig Latin for the querying process and big data analytics using Pig Latin[3].

[12] proposes two aggregation operations for extracting a schema from a document-oriented database. Their approach took almost 1 hour to process 17 million documents. [13] proposed hybrid (path-node) labeling to map document data from XML to a relational database. Their approach applies node-labelling to annotate positions of nodes in a data model and adopts path-labeling to boost performance of the algorithm.

[1] extracts an XML schema by using a tokenization process that uses a stemming algorithm to reduce words to stem or root forms. Then, they correlate the data linguistically by using WordNet to enable integration between multiple data sources and generate schemas without domain-specific knowledge.

[3] extracts the schema using the MapReduce paradigm from HDFS. They define two-step processing using the Map() function and the Reduce() function in this model. The MapReduce system runs on a cluster type platform, then automatically parallelizes the processes by cutting the processes into sub-processes where each will be assigned to a node (the map function runs on the cluster machine). Some of the results will be sent to a reducer (reducing the tasks performed on the cluster machine) to produce the data schema.

D. Google Cloud Platform

Google Cloud Platform (GCP) is the common name for cloud services that Google provides with "pay as per use" services, flexible infrastructure, other services such as high-level data analysis, machine learning that supports Google search and Gmail. The services provided by GCP, one of which are Cloud Storage, App Engine, and BigQuery [14].

1) Cloud Storage: Cloud Storage is a service for storing objects in Google Cloud. The Objects will be stored in a container called a bucket. All buckets will be associated with projects that can be grouped under an organization.

Once a project and a Cloud Storage bucket have been created, objects can be uploaded and downloaded from the bucket. Cloud Storage also provides services that can grant permission to make data accessible to specified members or accessible to everyone on the public internet (such as hosting a website).

Cloud Storage offers durable and highly available object storage for both structured and unstructured data. For example, the data are log files, database backup and export files, images, and other binary files.

2) App Engine: Create and deploy applications in a fully managed platform. Quickly scale application from zero to planetary without worrying about managing the underlying infrastructure. With "zero servers" management and zero-configuration deployment, developers focus solely on building advanced applications with no management costs. App Engine makes it easy for developers to be more productive and dynamic by developing popular languages and a wide range of developer tools.

3) BigQuery: Storing and querying large data sets can be time-consuming and expensive without the proper hardware and infrastructure. BigQuery is an enterprise data warehouse that solves this problem by enabling super-fast SQL queries using the processing power of Google's infrastructure. Move the data to BigQuery, and the data will be handled by default. Accessing the projects and data can be controlled based on need, such as granting others access rights to view and query data stored in the data warehouse.

[3] https://pig.apache.org/docs/r0.17.0/basic.html

BigQuery can be accessed using the Cloud Console, the bq command-line tool, or making calls to the BigQuery REST API using various client libraries such as Java, .NET, or Python. Various third-party tools can interact with BigQuery, such as visualizing or loading data. To getting started with BigQuery, there is no need to use any resources, such as disks and virtual machines.

E. The Format of Files

1) XML: XML stands for eXtensible Markup Language, which is a markup language much similar to HTML. XML was created to store and transport data and to be self-descriptive [15]. An XML file contains a formatted dataset intended to be processed by a website, web application, or software program. XML files can be thought of as text-based databases.

2) RDF: The Resource Description Framework (RDF) is a language standardized by the W3C to represent every piece of information about a Web resource [16]. An RDF file is a document written in the Resource Description Framework (RDF) language, which represents information about resources on the web. It includes information about a website in a structured format called metadata. RDF files may consist of a site map, an updated log, page descriptions, and keywords.

3) JSON: JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is simple for humans to read and write and for machines to parse and generate. JSON is a text format that is independent of any programming language. It uses a language style usually used by C-family programmers, including C, C++, C#, Java, JavaScript, Perl, Python, etc. Because of these characteristics, it makes JSON ideal as a data-exchange language [17].

III. OUR APPROACH

We propose the usage of Google Cloud Platform to extract a schema from a document. The components of our approach are as follows: Cloud Storage, App Engine, and BigQuey, flask module, json module, and three python modules (load_data_from_dblp(), load_data_from_makg() and load_data_to_bigquery()).

Based on Fig. 3, our approach works as follow: (1) collecting documents, (2) preprocessing documents, (3) uploading documents to Cloud Storage, (4) extracting and transforming schema in App Engine, and (5) loading data and schema from App Engine to BigQuery.

The complete steps of our approach are presented in Fig. 3. The first step is to collect or download documents from the internet. The type of documents that we collect is XML and RDF. After being collected, the next step is to preprocess the documents. In the document preprocessing step, we use EasyRDF[4] and Code Beautify[5] to convert the document from its raw format into JSON.

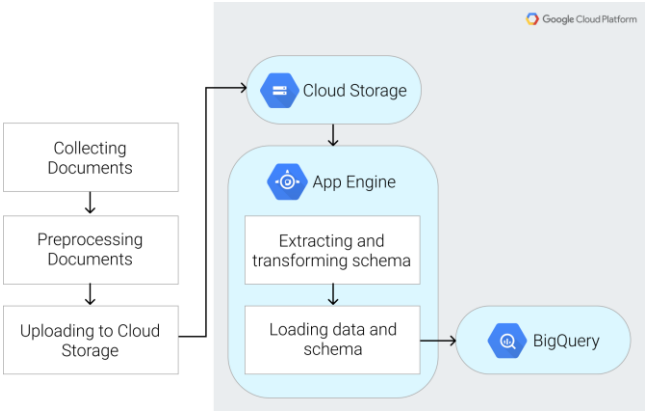

Fig. 3 Steps for extracting schema from a document

[4] https://www.easyrdf.org/converter
[5] https://codebeautify.org/xmltojson

The third step is uploading the documents to Google Cloud Storage which supports unstructured data. The information stored here is the document that encodes and encapsulates the data in a semi-structured format (such as JSON). A document of different structures can be stored in the same set. As in Fig. 4, a document consists of fields and their associated values, which are considered a hierarchy of elements that can be atomic values or composite values.

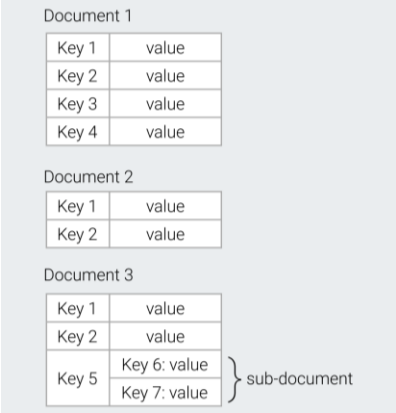

Fig.4 A structure of a document-oriented database

We divide the fourth step into more detailed steps as follows: (i). analyzing the characteristics of different documents; (ii). extracting all the keys and values from the documents; (iii). sorting the key and values in ascending mode. We call this step (iii) as transform 1; (iv). removing duplicate keys and values. We call this step (iv) as transform 2. We implement these four sub-steps using python. The result of step 4 is a generic schema (pair of keys and values) that encompasses all the keys and values from all the documents. This generic schema will be loaded together with the documents to a data warehouse. For keys that have the same meaning but different labels, we make a script to detect it, and then we give a new common label for both keys.

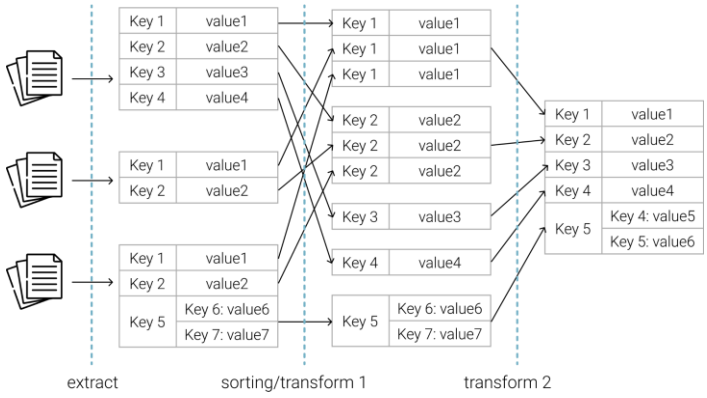

Fig. 5 Extracting and transforming schema in App Engine

Finally, using load_data_to_bigquery(), we pass this schema and data into Big Query for easy analysis and gain insight.

IV. Results And Discussion

For evaluating our approach, we use Database Systems and Logic Programming (DBLP) and Microsoft Academic Knowledge Graph (MAKG) documents. DBLP and MAKG are online references for open bibliographic information about scientific publications. We select ten documents for each dataset.

The data structures (set of keys and values) of a document from each dataset have different characteristics, as shown by Fig. 6.

dblp1.json

```
{
   “dblp”: {
      “article”: {
         “author”: [
            “Muhammad Bagus Andra”,
            “Tsuyoshi Usagawa”
         ],
         “title”: “Improved...”,
         “pages”: “70758-70774”,
         “year”: 2021,
         “volume”: 9,
         “journal”: “IEEE Access”,
         “ee”: “https://doi.org/...”
         “url”: “db/journals/...”
      }
   }
}
```

makg1.json

```
{
   “Description”: {
      “type”: “”,
      “rank”: 24507,
      “title”: “The Change...”
      “publicationDate”: “2017-01-01”,
      “publisher”: “ROK Army Military...”,
      “appearsInJournal”: “”,
      “issueIdentifier”: 143,
      “startingPage”: 439,
      “endingPage”: 477,
      “citationCount”: 0,
      “estimatedCitationCount”: 0,
      “created”: “2018-06-01”
   }
}
```

Fig. 6 An example of a document of DBLP and MAKG dataset

In Fig. 6, the DBLP and MAKG documents have different hierarchical depths. The deeper the hierarchy, the easier it is to model the details. Furthermore, the document with a deeper hierarchy has a better organization of the keys and values when it comes to multiple values (please see the author key in dblp1.json as an example).

We also can present the document from both datasets in a tree format (please see Fig. 7). In Fig. 7, the root of dblp.json, which is in hierarchy-0, is "dblp". Going deeper, in hierarchy-1, an "article" node has a node child of the needed information placed at hierarchy-2. Unlike makg.json, which has the root "Description" in hierarchy-0 and the nodes of the necessary information at hierarchy-1. Thus, the depth of a DBLP document is greater than the depth of a MAKG document.

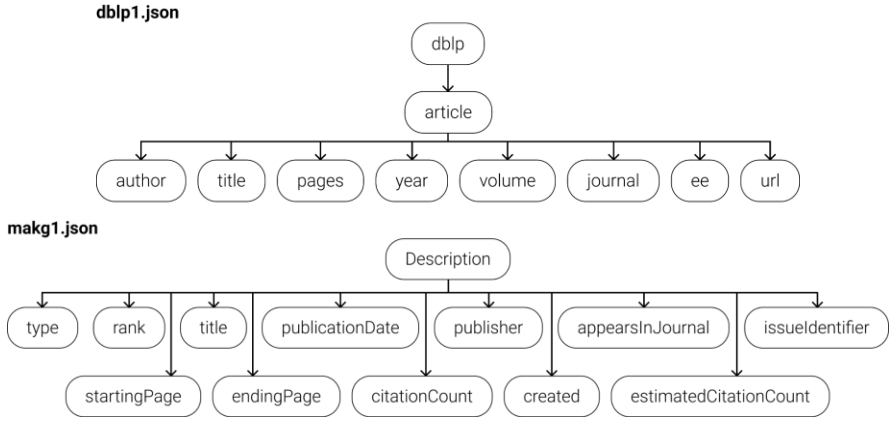

Fig. 7 A tree version of a document of DBLP and MAKG dataset

To evaluate the quality of the schema produced by our approach, we use a third software application called Oxygen[6]. Oxygen is a simple XML editor that can validate a schema of a document. It took 0.49 seconds for Oxygen to validate the generic schema produced by our approach, and Oxygen reports that our schema consists of 18 keys and 20 values (per row) is valid.

Validating the document schema by selecting "Validate with" in the "Validation" dropdown menu. After the "Validate with" dialog box opens, the "URL" section is filled with the schema location that has been

[6] http://oxygenxml.com/

downloaded from BigQuery, and "Schema type" is filled with the schema data type, which in this case is JSON, then click OK.

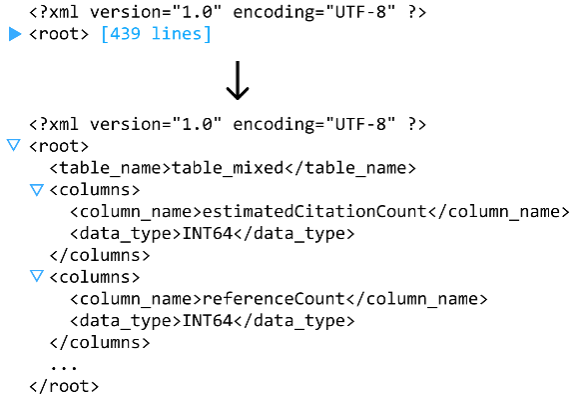

Fig. 8 Illustration expanding all tags by Oxygen

Oxygen validates the schema file by expanding all tags (like Fig. 8) on the included module to validate the entire schema hierarchy. Validation issues are highlighted directly in the editor, making it easy to find and fix any issues.

To evaluate the usefulness of our approach, we asked ten developers who understand the concepts of the JSON and SQL languages to formulate queries with or without looking at generic schemas. We want to test whether schema helps and speeds up developers in preparing queries or not. Each developer is given a document containing a questionnaire, instructions, and expected results in this test scenario. There were two experiments carried out. In the first experiment, developers were given ten DBLP datasets and ten MAKG datasets and asked to formulate queries using the programming language they were familiar with. And then, in the second experiment, the developer is given a BigQuery GCP account to write queries by observing the generic schema and viewing the datasets in the data warehouse. The system will automatically record query writing on BigQuery. The queries that need to be formulated are in Table II:

TABLE II
FORMULATED QUERIES

1. Make a list of titles starting with the author's name 'Achmad'

dblp6.json

```
...
“author”: [
   “Dwiza Riana”,
   “Achmad Nizar Hidayanto”,
   ...
]
“title”: “Integrative Factors of E-
Health Laboratory Adoption: A Case of
Indonesia.”
...
```

dblp10.json

```
...
“author”: [
   “Nita Arryani Sari”,
   “Achmad Nizar Hidayanto”,
   ...
]
“title”: “Impact of enterprise
architecture management on business
benefits through information technology
benefits in companies in Indonesia.”
...
```

2. Make a list of titles that have the word 'Indonesia'

dblp1.json

```
...
“title”: “Improved Transcription and
Speaker Identification System for
Concurrent Speech in Bahasa Indonesia
Using Recurrent Neural Network.”
...
```

dblp2.json

```
...
“title”: “Spatiotemporal Analysis of
COVID-19 Spread with Emerging Hotspot
Analysis and Space-Time Cube Models in
East Java, Indonesia.”
...
```

3. Make a list of doi and ee

4. Make a list of publishers

dblp1.json

```
...
“ee”: “https://doi.org/10.1109/
ACCESS.2021.3077441”
...
```

dblp2.json

```
...
“ee”: “https://doi.org/10.3390/
ijgi10030133”
...
```

makg1.json

```
...
“publisher”: “ROK Army Military History
Institute”
...
```

makg3.json

```
...
“publisher”: “Hume Society”
...
```

5. Look for data with the author's name 'Ahmad R. Pratama'

dblp4.json

```
{
  "dblp": {
    "article": {
      "author": "Ahmad R. Pratama",
      "title": "Fun first, useful...",
      "pages": "1737-1753",
      "year": 2021,
      "volume": 26,
      "journal": "Educ. Inf. Technol.",
      "number": 2,
      "ee": "https://doi.org/10.100...",
      "url": "db/journals/eait/..."
    }
  }
}
```

6. Search data with publisher 'Elsevier'

makg10.json

```
{
  "Description": {
    "type": ["",""],
    "rank": 21993,
    "doi": "10.1016/...",
    "title": "A statistical study...",
    "publicationDate": "2015-01-08",
    "publisher": "Elsevier",
    "appearsInJournal": "",
    "volume": 635,
    "startingPage": 45,
    "endingPage": 49,
    "referenceCount": 38,
    "citationCount": 2,
    "estimatedCitationCount": 2,
    "created": "2016-06-24"
  }
}
```

7. Look for data with a volume of more than 30

dblp5.json

```
{
  "dblp": {
    "article": {
      "author": [
        "Joan Santoso",
        "Esther Irawati Setiawan",
        "Christian Nathaniel Purwanto",
        "Eko Mulyanto Yuniarno",
        "Mochamad Hariadi",
        "Mauridhi Hery Purnomo"
      ],
      "title": "Named entity...",
      "pages": 114856,
      "year": 2021,
      "volume": 176,
      "journal": "Expert Syst. Appl.",
      "ee": "https://doi.org/10...",
      "url": "db/journals/eswa/..."
    }
  }
}
```

8. Look for the author's name from the article entitled 'Integrative Factors of E-Health Laboratory Adoption: A Case of Indonesia'

dblp6.json

```
...
“author”: [
   “Dwiza Riana”,
   “Achmad Nizar Hidayanto”,
   ...
]
“title”: “Integrative Factors of E-
Health Laboratory Adoption: A Case of
Indonesia.”
...
```

9. Look for data with the author's name 'Muhammad Bagus Andra'

```
dblp1.json
{
  "dblp": {
    "article": {
      "author": [
        "Muhammad Bagus Andra",
        "Tsuyoshi Usagawa"
      ],
      "title": "Improved Transcript...",
      "pages": "70758-70774",
      "year": 2021,
      "volume": 9,
      "journal": "IEEE Access",
      "ee": "https://doi.org/10.110...",
      "url": "db/journals/access/..."
    }
  }
}
```

10. Look for data with the journal name 'Data'

```
dblp3.json
{
  "dblp": {
    "article": {
      "author": [
        "Dana Indra Sensuse",
        "Viktor Suwiyanto",
        "Sofian Lusa",
        "Arfive Gandhi",
        "Muhammad Mishbah",
        "Damayanti Elisabeth"
      ],
      "title": "Designing Knowledge...",
      "pages": 48,
      "year": 2021,
      "volume": 6,
      "journal": "Data",
      "number": 5,
      "ee": "https://doi.org/10...",
      "url": "db/journals/data/..."
    }
  }
}
```

With a different schema for each dataset document, formulating a query without a schema requires longer because the developer needs first to find or guess the intended key, especially when the key is not found in the opened dataset document. We report the results of the first and second experiments in the "Without schema" and "With schema" columns.

TABLE III
THE AVERAGE TIME TO WRITE QUERIES

	Without schema	**With schema**
The average time to write 10 queries	51,41 minutes	15,86 minutes

From the results in Table III, ten queries were conducted by ten developers with categories "with schema" and "without schema". Based on the table above, the absence of a schema requires developers to search for attributes manually and results in an average time of 51,41 minutes. On the other hand, the schema makes it easier for developers to quickly understand and find the features/keys and values they are looking for, thereby reducing the query writing time by about 35,55 minutes. From this usefulness evaluation, we proved that our approach helps the developers formulate queries better and faster.

In producing a generic schema (as shown in Fig. 9), we notice some interesting findings that we need to report as follows.

(1). Keys that have different labels but similar meanings. The key "pages" in DBLP and the keys "startingPage" and "endingPage" in MAKG have different labels, but they are pointing to similar meanings. The key "pages" tells us the range of a paper, ex: "11-15". Whereas the value of the key "starting page" and "ending page" are "11" and "15" respectively. Therefore, for this case, we choose "pages" as the label in the generic schema due to its brevity.

(2). Keys that have different labels but the same meaning. The key "ee" in DBLP and the key "doi" in MAKG refers to the same purpose, which is a unique string that identifies where a resource is located on the internet [18]. For this case, we need to manually choose which label we will use in the generic schema.

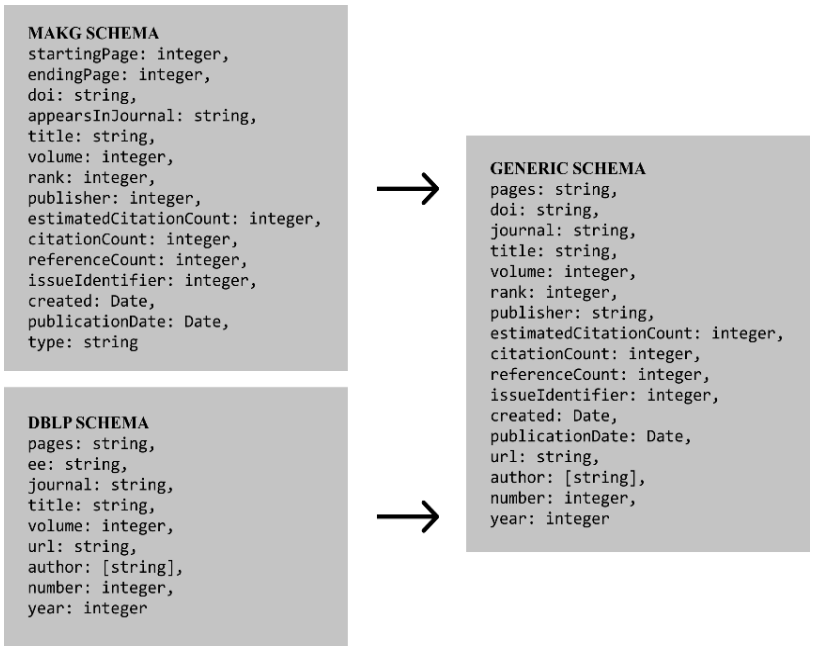

Fig. 9 A transformation schema

V. CONCLUSION

This paper proposes an approach that extracts data warehouse schema from a document-oriented database using Google Cloud Platform as a service. The proposed method consists of five main steps: collecting NoSQL documents, document preprocessing, storing documents in Cloud Storage, extracting and transforming schemas, and loading the data and schemas from App Engine to BigQuery data warehouse.

We evaluate our approach using DBLP and MAKG datasets. Although DBLP and MAKG have different data characteristics and depth, our approach successfully produces a generic and validated schema. We tested the usefulness of this schema to a group of developers, and we found out that the query writing process using schema is 35,55 minutes faster than writing without schema. It proves that the schema of a document-oriented database helps the developers understand and formulate the queries better and quicker.

There are two potential paths for future work: (1). automating and integrating the schema extraction process for document-oriented databases with a data warehouse, (2). creating a generic extraction framework for other types of databases, such as graph, column-oriented, or key values databases.

VI. DATA AND COMPUTER PROGRAM AVAILABILITY

Data can be accessed in the following site bit.ly/datasetdblpmakg and github.com/anurulistiqamah/lastpro.

REFERENCES

[1] A. A. Alqarni and E. Pardede, “Integration of data warehouse and unstructured business documents,” in *Proceedings of the 2012 15th International Conference on Network-Based Information Systems, NBIS 2012*, 2012, pp. 32–37. doi: 10.1109/NBiS.2012.59.

[2] E. Gallinucci, M. Golfarelli, and S. Rizzi, “Schema Profiling of Document-Oriented Databases,” *Information Systems*, vol. 75, pp. 13–25, Jun. 2018, doi: 10.1016/j.is.2018.02.007.

[3] S. Bouaziz, A. Nabli, and F. Gargouri, "Design a Data Warehouse Schema from Document-Oriented Database," in *Procedia Computer Science*, 2019, vol. 159, pp. 221–230. doi: 10.1016/j.procs.2019.09.177.
[4] M. I. Halim, "Penerapan Document Oriented Database (NOSQL) Dalam Pembuatan E-LIBRARY Universitas Pendidikan Indonesia Menggunakan Mongodb Dan PHP," 2016.
[5] S. Tiwari, *Professional NoSQL*. Indianapolis: John Wiley & Sons, Inc., 2011.
[6] A. Stevenson, *Oxford Dictionary of English*. USA: Oxford University Press, 2010.
[7] "Column-oriented DBMS," *Wikipedia*, 2021. https://en.wikipedia.org/wiki/Column-oriented_DBMS (accessed Aug. 07, 2021).
[8] K. Ibrahim Mohammed, "Data Warehouse Design and Implementation Based on Quality Requirements," *International Journal of Advances in Engineering & Technology*, vol. 7, pp. 642–651, 2014, [Online]. Available: https://www.researchgate.net/publication/330666318
[9] S. H. A. Aloush, "The Role of Data Warehouse in Decreasing the Time," *Australian Journal of Basic and Applied Sciences*, vol. 9, no. 5, pp. 216–219, 2015.
[10] P. S. Kumar, M. Antigopal, and S. Vetrivel, "Extract Transform and Load Strategy for Unstructured Data into Data Warehouse Using Map Reduce Paradigm and Big Data Analytics," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 02, no. 12, pp. 7456–7462, Jan. 2015, doi: 10.15680/ijircce.2014.0212030.
[11] H. Saradava, A. Patel, and R. Aluvalu, "A Survey on ETL Strategy for Unstructured Data in Data Warehouse using Big Data Analytics," 2016.
[12] A. A. Frozza, R. dos S. Mello, and F. de S. da Costa, "An Approach for Schema Extraction of JSON and Extended JSON Document Collections," Jul. 2018. doi: 10.1109/IRI.2018.00060.
[13] H. Zhu, H. Yu, G. Fan, and H. Sun, "Mini-XML: An efficient mapping approach between XML and relational database," May 2017. doi: 10.1109/ICIS.2017.7960109.
[14] "Apa itu GCP?," *Cloud Ace Indonesia*, 2021. https://id.cloud-ace.com/id/what-is-gcp-id/ (accessed Aug. 07, 2021).
[15] "XML Introduction," *W3Schools*, 2021. https://www.w3schools.com/xml/xml_whatis.asp (accessed Aug. 10, 2021).
[16] F. Manola and E. Miller, "RDF Primer," *W3C*, 2004. https://www.w3.org/TR/rdf-primer/ (accessed Aug. 10, 2021).
[17] "JSON," *Json.org*. https://www.json.org/json-en.html (accessed Sep. 10, 2021).
[18] "DOIs: What they are and how to cite them: Overview," *Montana State University*, 2021. https://guides.lib.montana.edu/doi/ (accessed Aug. 10, 2021).