SN Computer Science (2021) 2:325
https://doi.org/10.1007/s42979-021-00716-3

ORIGINAL RESEARCH

An Elastic Data Conversion Framework: A Case Study for MySQL and MongoDB

Tran Khanh Dang[1] · Ta Manh Huy[1] · Ly Hoang Dang[1] · Nguyen Le Hoang[1]

Received: 22 March 2021 / Accepted: 19 May 2021 / Published online: 5 June 2021

Abstract
Data nowadays are extremely valuable resource. However, data are created and stored in different places with various formats and types. As a result, it is not easy and efficient for data analysis and data mining which can make profits for every aspect of social applications. To overcome this problem, data conversion is a crucial step that we have to build for linking and merging different data resources to a unified data store. In this paper, based on the intermediate data conversion model, we proposed an elastic data conversion framework for data integration system. Besides, we also performed an experiment to evaluate our model using MySQL and MongoDB.

Keywords Data conversion · Data integration system · Data transformation · Open data

Introduction

With the development of technology, data is becoming an extremely valuable resource. Data is being created, analyzed and used in a massive scale in every modern system. As a result, data analysis and data mining are very essential in each aspect of social applications. Data will even be put to much better use if we combine the data from different sources into bigger, more informative data sets, making the data even more suitable for the analyzing and mining tasks. The integration of data is especially useful for solving current social problems [1, 2]. However, to make data integration feasible, data transformation is a crucial challenge that we have to overcome.

Data transformation can be described as a task that can flexibly convert data among different models and formats, thereby supporting the combination of multiple data sets from various sources to a unified one, in another word, a unified data set. The problem of data transformation is not easy, even when converting traditional data model like relational data model with few data sources that has simple structures. The transformation process usually requires the participation of human to understand the meaning of the data in each data source to solve the data ambiguity problems, including semantic and data representation ambiguity.

In the age of big data, the problem of data conversion becomes more and more challenging when data are not only heterogeneous, but are also produced continuously with enormous mass [3]. These following three main characteristics of big data are known through the notation "3V":

- Volume: Data sources not only contain a large amount of data but the number of data sources also becomes very large.
- Velocity: Data are continuously generated and changed over time.
- Variety: Data from many different sources have diverse and heterogeneous structures.

This article is part of the topical collection "Future Data and Security Engineering 2020" guest-edited by Tran Khanh Dang.

✉ Tran Khanh Dang
khanh@hcmut.edu.vn

✉ Nguyen Le Hoang
nlhoang@hcmut.edu.vn

Ta Manh Huy
tamanhhuy810@gmail.com

Ly Hoang Dang
lyhoangdang96@gmail.com

[1] Ho Chi Minh City University of Technology, Vietnam National University Ho Chi Minh City, 268 Ly Thuong Kiet, District 10, Ho Chi Minh City, Vietnam

Data transformation is an essential problem in many industries. For example, the traffic data integrated from the bus black boxes and the cameras on the road provide a comprehensive view of the traffic situation of the city. When combining these information with data on population such as population density and distribution, the management agencies and related departments will be able to make appropriate decisions and policies such as traffic flow, reconstructing and establishing traffic infrastructure, or navigating traffic to avoid traffic jams. The problem is that departments often store data with completely different models and formats. Hence, data transformation is an indispensable step in the integration, analysis, and decision-making process. In the U.S., transport agencies rely on large amounts of data to support everyday tasks such as planning, designing, and construction [4]. Therefore, these agencies also need to gather and exchange a lot of information. The access speed together with the accuracy and consistency of the information from these different platforms and targets lead to the problem of data conversion. In addition, the converted data sets can be combined together into a unified dataset and through data mining process, the unified dataset can bring many benefits to data analysis and management applications as well as can provide potential and optimal value [5–7]. Furthermore, this combined dataset is a rich resource in making predictions and supporting decision making. Data now can be collected and integrated to store and manage in data centers for a variety of purposes (Fig. 1).

However, the data transformation's main challenge is that the structure and format diversity of the various data source. Hence, it is necessary to do research and propose a data standard format to support storage in data centers and propose a framework supporting data transformation before integrating them into data centers. This research direction is also one of the research trends on Information Technology for the Ho Chi Minh City in the period of 2018–2023. In this paper, we propose a novel data conversion framework for data integration system. The rest of this paper is organized as follows: some related works and researches are discussed in the next section, and our proposed framework is explored in the subsequent section. In the penultimate section, we performed experiment and evaluation. The final section is about the summary and conclusion of our work.

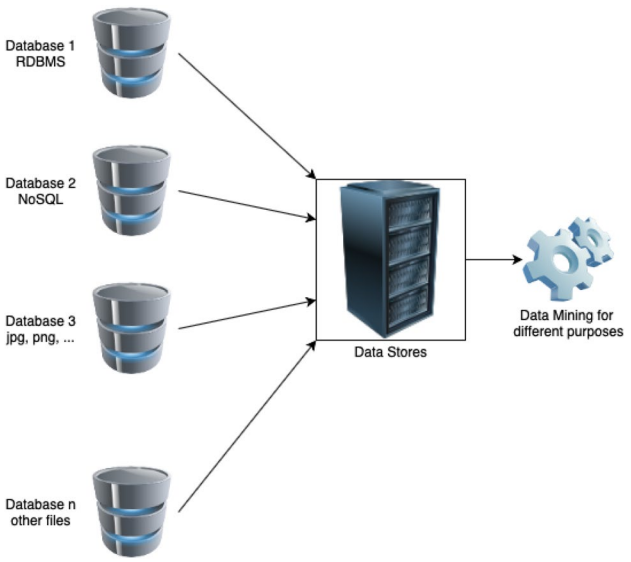


Fig. 1 Data stores stores the combined data sets-making it very valuable for mining and analyzing task

Related Works

Since 2010, there have been a lot of researches in the field of data conversion and the researchers have proposed some methods for data conversion. In 2013, Ivan et al. proposed a data transformation system based on a community contribution model [8]. As depicted in Fig. 2, the data shared on the *publicdata.eu* portal included data from many different organizations of various formats. The system would then make initial mappings, then the community could contribute by creating new mappings, re-editing existing mappings, transforming the data, and using the data. The accuracy in data conversion would be improved over time with the contribution of the community.

In 2015, Rocha et al. proposed a method to support the migration of data from relational databases management system (RDBMS) to NoSQL [9]. This method included 2 main module: data migration and data mapping.

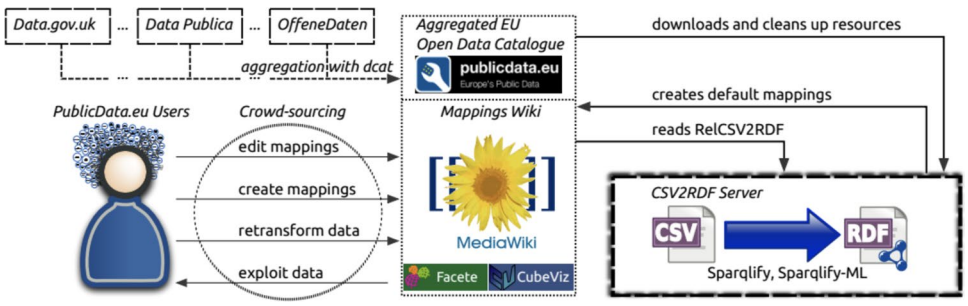


Fig. 2 Data transformation system based on a community contribution model [8]

- The data migration module's (Fig. 3) responsibility included automatically identifying all elements from the original relational databases (e.g. tables, properties, relationships, indexes, etc.), creating equivalent structures using the NoSQL data model and exporting the data to the new model.
- The data mapping module (Fig. 4) consisted of an abstract class, it was designed as an interface between the application and the DBMS. This module oversaw all SQL transactions from the applications, and translated these operations then moved to the NoSQL model that was created in data migration module.

Hyeonjeong et al. developed a semi-automatic tool for converting ecological data in Korea in 2017 [10]. The goal of this tool was to gather data in different formats from various research organizations and institutes specializing in environment in Korea and then convert to a shared standard ecological dataset. To accomplish this goal, the authors proposed 4 transformation steps as described in Fig. 5 including:

- Step 1: Data file and protocol selection: This step provided an interface that allowed users to select data from the source file and the corresponding protocol.
- Step 2: Species selection: The user chose which species in the data to be converted.
- Step 3: Attribute mapping: This step was responsible for mapping attributes from source data to normalized attributes defined in the protocol.

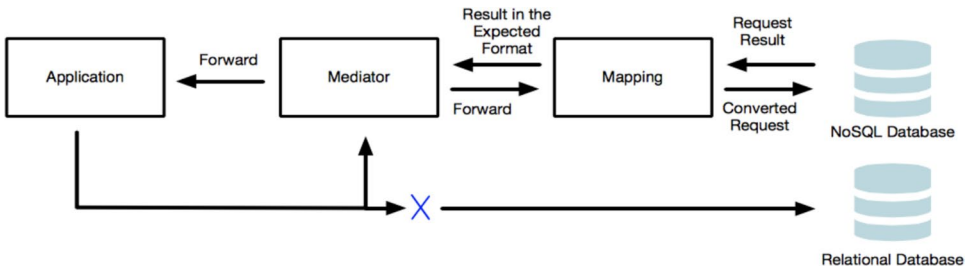


Fig. 3 The migration of data from relational databases RDBMS to NoSQL-data migration module [9]

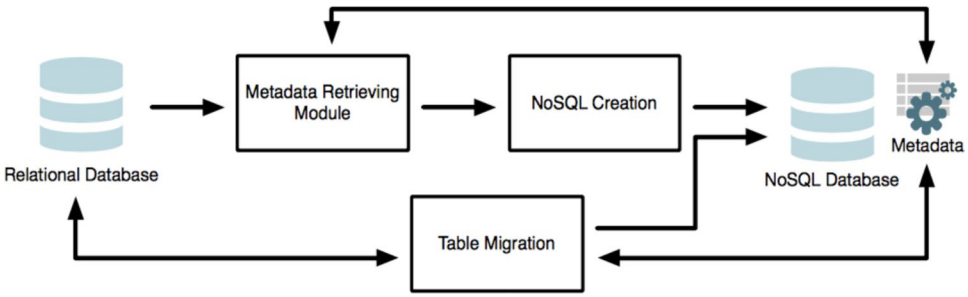


Fig. 4 The migration of data from relational databases RDBMS to NoSQL-data mapping module [9]

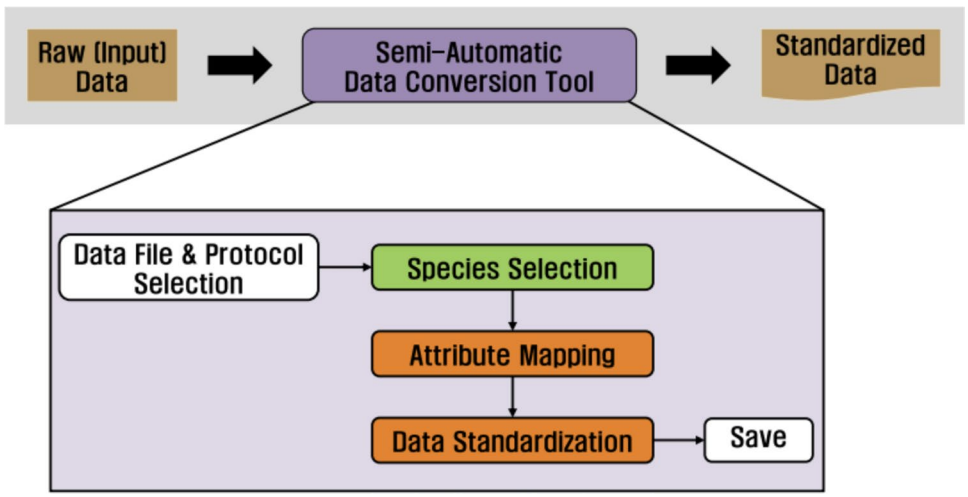


Fig. 5 Semi-automatic tool for converting ecological data in Korea [10]

- Step 4: Data standardization: This step bear the responsibility for converting mapped data to a shared standard.

However, this tool could only convert data for few species from the original data. Another limitation is that it only supported data sources stored in **.csv** format whereas the actual data are usually represented in many different formats.

In 2017, Milan et al. looked to the context of factory integration through the use of the data transformation toolkit for AutomationML (AML), an open standard XML-based data format for storage and exchange of technical information of the plant [11]. In this context, factory automation requires the participation and collaboration in a variety of fields from automation control, mechanical engineering, electronics, and software engineering. These domains all have different support tools, and the tools manipulate different data structures. Therefore, the authors proposed a model integrating these tools with AML using a process engineering transformation tool. This model converted the data described by the AML into the appropriate formats corresponding to the technical tools of different disciplines as depicted in Fig. 6. Although the model could work well, the input of the process could only be stored in AML standard.

In 2017, Luis et al. developed a data conversion framework to support energy simulation [12]. The goal of this framework was to convert data in different formats to enable communicating and interacting among different systems in an automated environment. This approach designed an intermediate component defined as the Interoperability Specification to enforce reciprocal interaction between two different data formats. Figure 7 illustrates the architecture of the interactive implementation. However, the study did not provide any more detail about their implementation.

Besides, data transformation solutions are also embedded in data integration systems. Dong and Srivastava in [13], based on traditional data integration architectures as depicted in

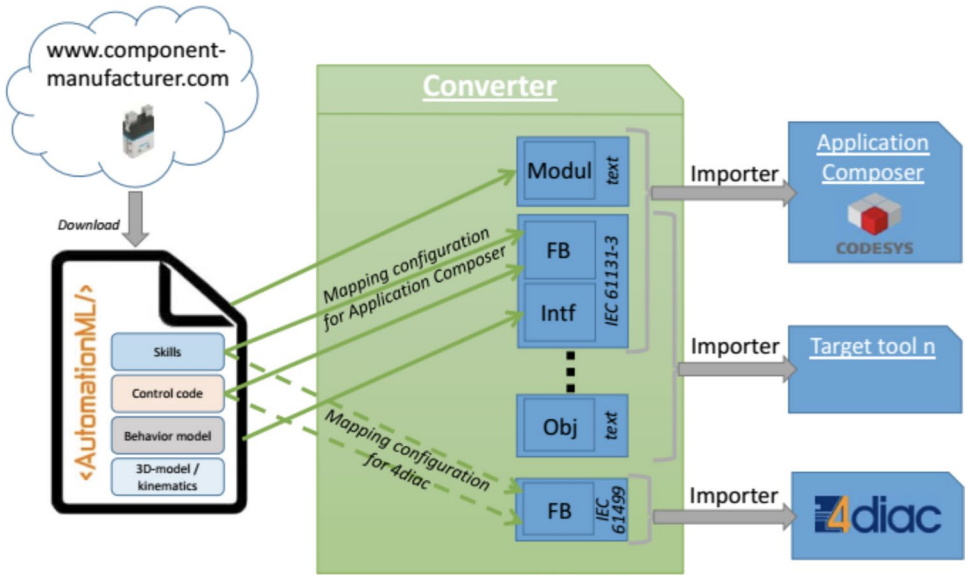


Fig. 6 Factory integration through the use of the data transformation toolkit for AutomationML [11]

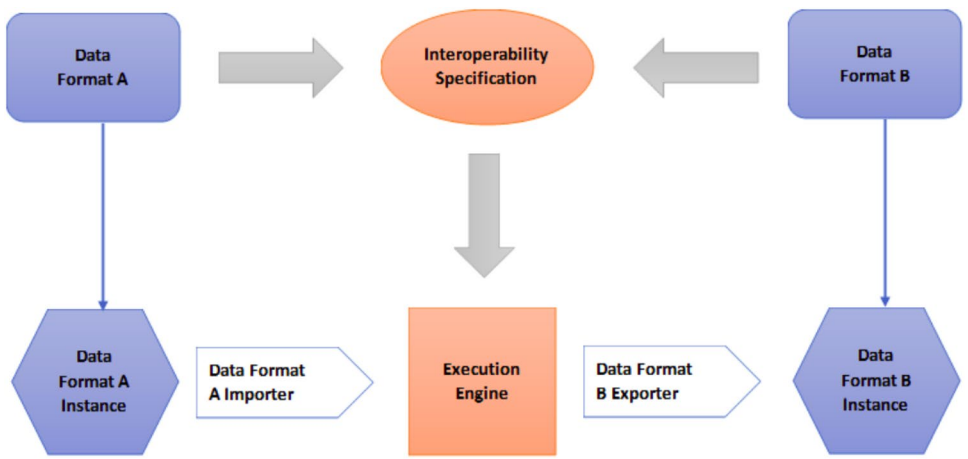


Fig. 7 Data conversion framework to support energy simulation [12]

Fig. 8, highlighted three main challenges of three main phrases of big data integration:

- Schema alignment: This phases solves semantic ambiguity challenges.
- Record linkage: This phases solves ambiguity challenges of data representation.
- Data fusion: This phases solves data quality challenges.

In another work of Marek et al. [14], the authors described different types of data inconsistencies, especially semantic inconsistencies. From there, the integration problem was classified according to two different challenges: low level and high level integration. In an attempt to reduce semantic heterogeneity, the authors used Semantic Web Technology for data integration and proposed architectural model called Semantic Big Data Historian (SBDH) with four main components as follows:

- Data acquisition layer: This layer collected data from sensors, additional internal data sources, or from external data sources. The problem of heterogeneity of platforms related to different systems would be solved in this layer.
- Data transformation layer: This layer converted data into integrated semantic form based on the proposed semantic network (SHS Ontology). This class could also correct the corrupted data if necessary. Semantic inconsistencies would be resolved in this class.
- Data storage layer: This layer played the role of data storage based on three systems: 4Store1, CumulusRDF, Hadoop and Jena Elephas.
- Analytic layer: This layer provided direct access to the storage layer for compost data analysis or user query processing. Selected analytical framework options included: KNIME, Mahout.

Knoblock and Szekely developed the Karma system, an integrated data system in the cultural heritage domain [15]. This system integrated data with high data heterogeneity from different museums. The process is described through four main stages:

- Data import phase: The data from any different source including databases, spreadsheets, or web services provided in XML or JSON format would be imported into the system.
- Data cleansing and normalization phase: In this phase, unusual data components and normalizing the data according to similar formats of related sources would be identified.
- Modeling phase: Semantic description of each resource would be created.
- Phase integration: The data would be converted into a single format using a description on semantics and data integration in an unified framework.

However, the above integrated system only focused on integrating the data source at the schema level, not the data link problem. Moreover, this system only considered the data heterogeneity factor (Variety), while the other two factors of big data, which are data volume (Volume) and the rate of data generation (Velocity), have not been mentioned.

There have been some studies related to data transformation and integration in Vietnam like the PhD thesis on integrating data models in the data center of oil and gas industry in Vietnam, the Master thesis on geographic data conversion tools integrated into GIS, or workshop on health data integration for management of smart health. In general, these studies focused on the problem of data integration in a specific field.

In the industry, there are also many products and tools for data conversion and integration. Information Builders launched the iWay Big Data Integrator that provided a modern approach to the conversion, integration and management of data based on the Hadoop platform [16]. Microsoft Corporation also has SQL Server Integration Services (SSIS) products with services that are able to extract and transform data from various data sources such as XML, files, and relational data sources, and load the data into one or more data storage [17]. Furthermore, Talend provides tools for big data integration and transformation solutions [18]. However, these tools perform the transformation of data directly through user interaction without the standard conversion data specification.

In Vietnam, the problem of data conversion has not been given adequate attention. Currently, almost data sources are stored individually at different departments, branches. Therefore, utilizing the value of this data source is very limited because it is difficult to combine data since each place often stores data in different formats and models. Although there are data centers in the infrastructure; however, these centers have not really combined data to create a unified data source to serve the needs of data mining and data analysis.

In the next section, we proposed a framework that converts data from many different sources and formats into a

Fig. 8 Challenges for big data integration [13]

common format called standard data format. Furthermore, we proposed a data conversion framework based on this standard one.

Proposed Framework

Approaching Methods for Data Conversion

The data conversion methods can be divided into two categories: direct conversion and intermediate conversion.

- Direct conversion: The data are converted directly from source to target format. This conversion method is the most popular conversion method due to its simplicity and ease of implementation. However, this method is only effective when the number of source and target formats is small. As this number increases, the complexity of the system increases rapidly (Fig. 9).
- Intermediate conversion: Data will be converted to intermediate data format and these data will be converted to the format that the user wants. This method has the advantage that it will reduce the complexity of the system and make the system much more easy to expand (Fig. 10).

Most studies and works mainly use the direct conversion model [10, 12]. The reason for the popularity of this model is due to the goal of the authors' projects: the need to convert data from one or several specific forms to one or

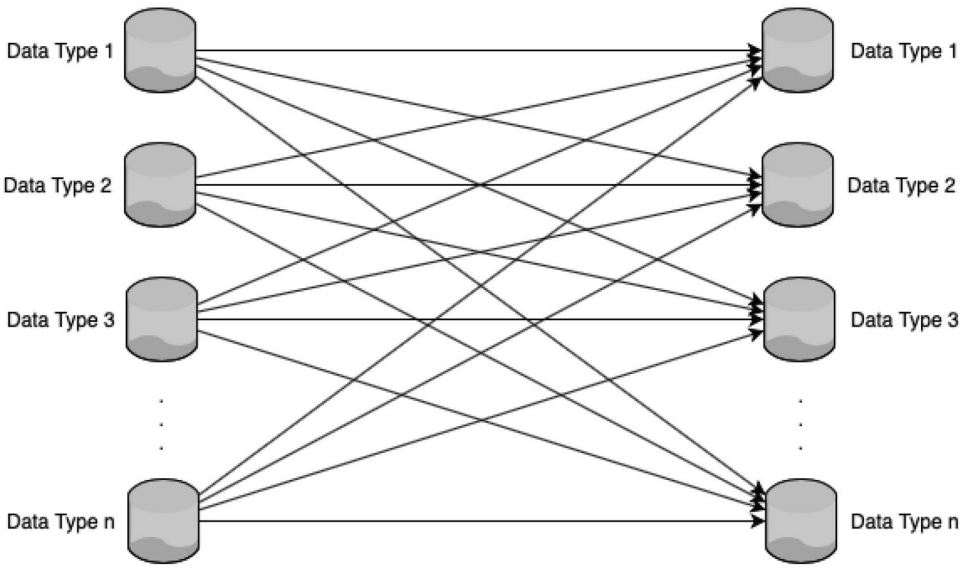


Fig. 9 Direct data conversion

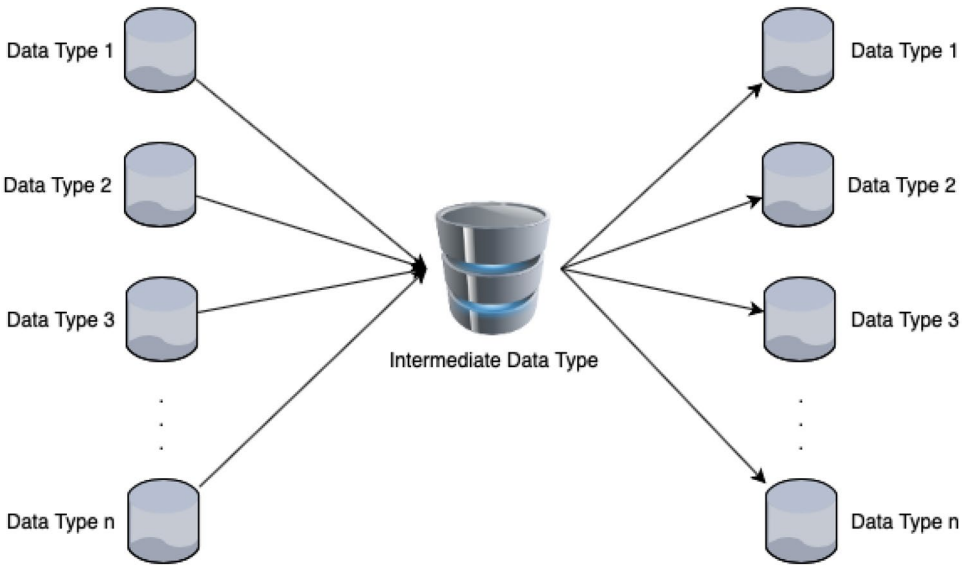


Fig. 10 Intermediate data conversion

several specific forms. However, with the goal of designing an extensible system that can work with a variety of input and output formats, the direct transformation model creates a lot of complexity when adding more data types for the system. In the direct conversion model, if we add a new input format to the system including n outputs of the system, it is required to manually programmatically extend n functions.

However, if the intermediate conversion model is used, only one additional function is needed for the system. Therefore, the direct conversion model is not suitable for the needs of a large and extensible data conversion system. In this paper, we used the intermediate conversion for our proposed framework. For this type of approaching method, one of the most significant problems is finding the most suitable intermediate data type for the system.

Framework Components

The framework contains six components (Fig. 11): I/O module, Data Stores module, Schema Detection module, Schema Conversion module, Schema Data module, and Validation module. All six components handle different processes of the system, making the conversion framework feasible.

Of the aforementioned six components of our system, the following three components are not directly involved in the conversion process: I/O module, Data Stores module, and Validation module. I/O module, as its name suggested, handles the input and output processes of the entire system. Data Stores module is responsible for the saving and storing data of the system. Validation module is used to check the input to ensure the data are convertible and validate the correctness of output before sending to user.

The remaining three components are responsible for transforming the data, these components are Data Stores module, Schema Detection module and Schema Conversion module. While Schema Detection module and Schema Conversion module both work with the schema of data, their responsibilities are quite different. Schema Detection module's goal is to detect and recognize the structure and schema of data. After the schema of the data has been identified by Schema Detection module, Schema Conversion module will convert the original schema into the target data format. Data Conversion module will then create a mapping for converting each original data record to target data format.

The Working Flow of the System

The data conversion system performs can be divided into two phases or processes: input-to-data-storage phase and data-storage-to-output phase. Figure 12 describes these two processes, the big arrows represent the flow of input process while the small arrows represent the flow of output process. The first phase is described as follows:

1. Input user's data: In this step, the data will be provided by the user to the system through the open data portal (I/O module).
2. Validation: The data user provide will be checked by Validation module whether it is suitable for converting or not. If it is suitable, a back-up file will be created then saved to the corresponding storage. If it is not, process will stop.
3. Schema detection: This step plays a core role in the whole process. In this step, Schema Detection module will read input data and detect its schema.

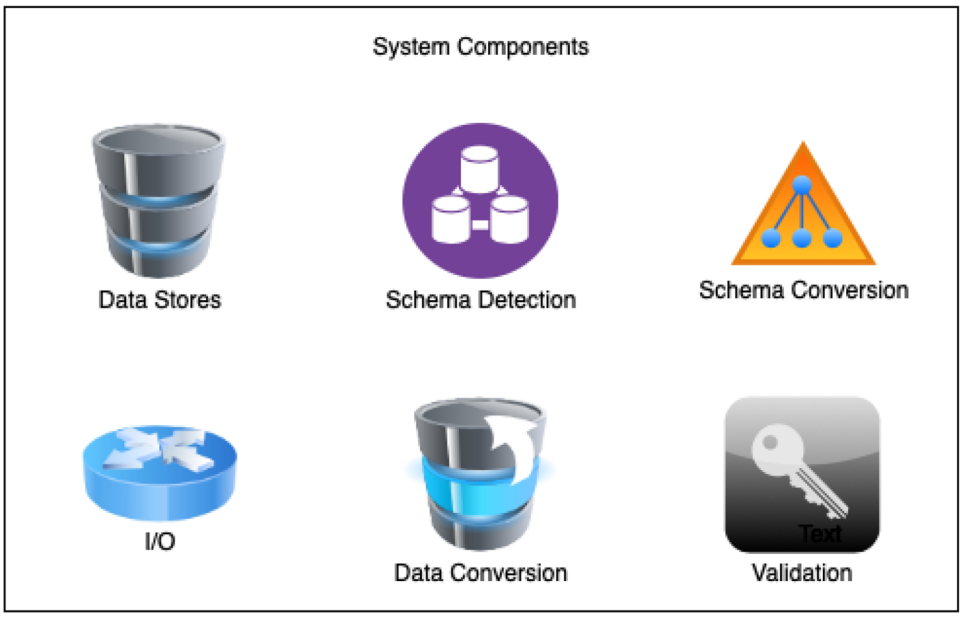


Fig. 11 Components of framework

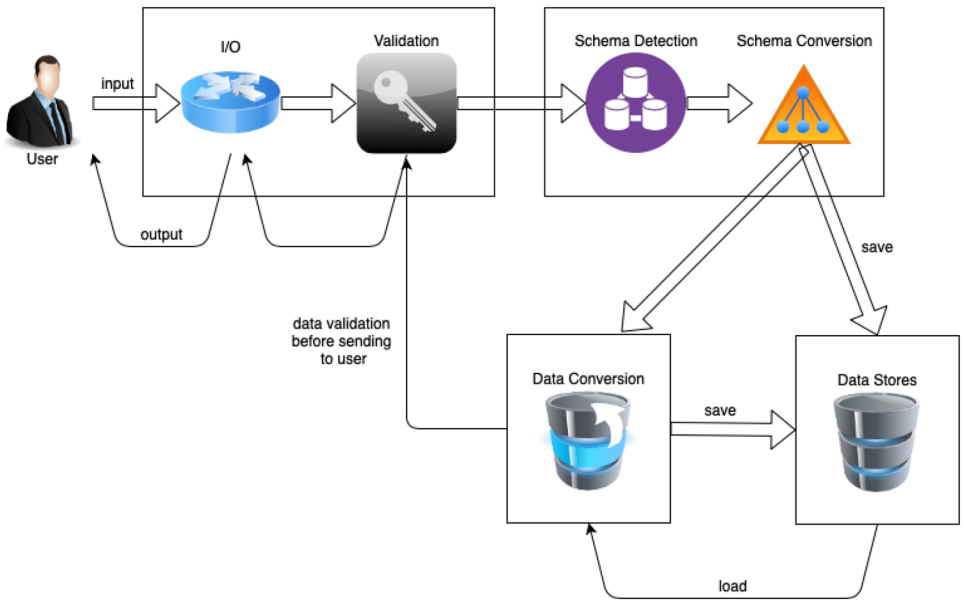


Fig. 12 Work flow of the system

4. Schema conversion: Schema Conversion module will convert data schema into intermediate format schema.
5. Data conversion: Data Stores module creates a mapping from the detected schema to the converted schema, the whole dataset will be converted through this mapping to intermediate data format.
6. Saving data to data stores: Once the conversion is finished, both the schema and the converted data will be saved in the data stores.

The second phase can be describe as follows:

1. Loading data from data stores: When receiving demands from users, the system will find the required datasets and load to Data Conversion module.
2. Conversion: Schema Conversion module will convert intermediate schema to target format schema and Data Conversion module will convert loaded data into destination format.
3. Validation: The correctness of final converted data will be validated (by comparing with saved schemas) before sending back to users.
4. Exporting data: Converted data to users will be exported and presented to the user through I/O module.

Framework Architecture

The overall structure of the data transformation framework is shown in Fig. 13.

I/O Module

The job of the Input/Output module is to receive and return data to the users. Because of its jobs, the I/O module is an interaction point between system and users. It is essential to communicate with the users through an intuitive interface. Therefore, we used an open data communication portal for this I/O module, because the open data portal will make the users' interaction with our framework much more convenient. This design will also help the system become more flexible and extensible. There is two ways a user can use to input data into the system: through a direct connection to user database system or through raw uploaded files. For some large or big data, there must be a module to ensure the integrity of the data.

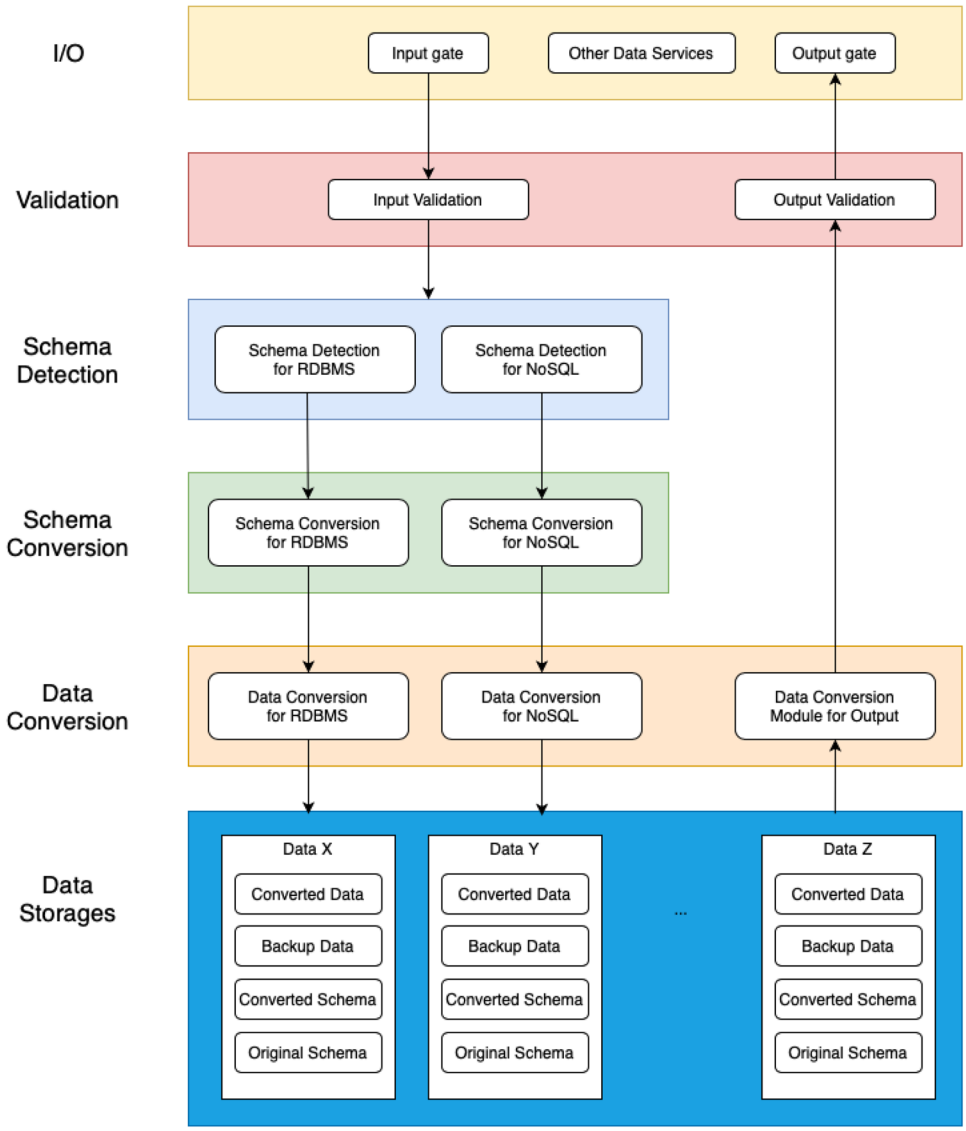
Validation

Every data must be checked before inputting to the system or outputting to users. For input data, validation module should:

- Ensure that data are of some specific formats that can be converted.
- For a set of files such as images and videos. Validation module should check and ensure the data are harmless to the system.

Before sending output data to users, the Validation module should:

Fig. 13 Framework architecture



- Ensure the output data are similar to the original data.
- Ensure that current user having the right to access the output or part of the output.

Schema Detection

This component is responsible for extracting and detecting the schema of the data provided by the user. According to the design of the framework, input data will include two types of data: structured data and NoSQL data. Therefore, the Schema Detection module must be able to handle both structured data and NoSQL data.

Schema Conversion

This module will convert the detected schema into the target data formats. This step requires the schema conversion module to be able to understand and recognize the meanings of original schema.

Data Conversion

This component is responsible for transforming the data according to the schema outlined in the previous step. Therefore, this component is only active when the schema-related modules have finished. The input of this component is the schema converted by the Schema Conversion module and the original user data. The output is the data after converted.

As shown in Fig. 13, this component also converts intermediate data in data stores back to destination formats. This is an inverted process but much more simple.

For big data, this process will take a lot of time. Parallel processing with big data frameworks such as Hadoop or Spark will make this task faster and more efficient.

Data Stores

For each input data, the data stores will save these following about the data:

- Backup data: Backup data are the original dataset. This is used for backup in case users need the original data.
- Converted data: Converted data are the data that have been converted into intermediate data format.
- Original schema: The schema acquired from Schema Detection module. This is used for validating process to ensure the output data are similar to the original data schema.
- Converted schema: The schema acquired from Schema Conversion module.

Furthermore, the data stores must satisfy these following conditions:

- Systematic data storage: Data Stores should have systematic data storage to help users creating, storing, searching data accurately and quickly.
- Data safety: Data stores must ensure that data integrity and security is one of the most important concerns in the public data storage.
- Data stores can allow simultaneous access of multiple users on data.

Experiments for MySQL and MongoDB

Intermediate Data Type

The framework is designed to use an intermediate data type (or standard data format) in the transformation process. In fact, this standard format of data will also be stored in the storage block, for later needs and purposes. This data standard format must meet the following criteria:

- Flexible structure: The standard format structure must be flexible enough to express data of other formats without losing their distinct characteristics.
- Efficient storage: The standard format storage size on disk should be as small as possible but the storage size could be increased to accommodate other criteria.
- Good scalability: The standard format should scale well when the size of data grows.

In [19], the authors have confirmed that to satisfy these above standards, only formats such as JSON, XML and BSON meet the needs since traditional database models would not be suitable for flexible structure. Moreover, among these data types, JSON seemed to be the best choice due to its several advantages (Table 1).

Therefore, in this experiment, we used JSON as the intermediate data type with these benefits:

- Require small storage: The data that use JSON format all have reasonable storage size on disk.

Table 1 Comparison between XML, JSON and BSON [19]

	XML	JSON	BSON
Number of related repositories on GitHub	About 95,000 (Oct 2020)	About 240,000 (Oct 2020)	About 900 (Oct 2020)
Number of questions on StackOverflow	About 0.6% of questions on Stack Overflow in 2019 [20]	About 1.5% of questions on Stack Overflow in 2019 [20]	No data available
Data streaming supported	Full supported such as XMPP, QuiX-Schematron, XLTL 3.0	Full supported such as Jackson_ (API), jq, logstash, ldjson-stream	Few
The format this format is based on	SGML	None (But based on JavaScript syntax)	JSON
The formats based on this format	YAML, Fast Infoset, SOAP, XML-RPC, Efficient XML Interchange (EXI)	YAML, BSON, Ion, Smile, CBOR, Mes-sagePack, Extensible Data Notation (EDN)	None
The standardization of the format	Standardized	Standardized	Not Standardized
The standard APIs of the format	DOM, SAX, XQuery, XPath	Clarinet, JSONQuery, JSONPath, JSON-LD	No Standard APIs
Notable libraries of the format	JSON_checker, YAJL, json-c, json-parser, JSONKit, JSONUtil, json2.js	DOM4J, StAX, JDOM, xml2js, libxmljs, sax, fast-xml-parser, xml-stream	Libbson, Mongo-GLib, bson4jackson, CookJson, PyMongo

- Widely used in a lot of projects: JSON is a mature format that is widely used in real-world applications.
- Fully supported community and technology: With its maturity, JSON format has a thriving support community and as well as a wide range of support libraries.
- Easy to learn for new users: JSON is easy for new users to learn and understand.

Initialization

We considered the current trend of using database management systems to choose the data formats for this experiment. According to statistics from DB-Engines [21] on March 2021, the first five positions in the list of most popular databases are Oracle, MySQL, Microsoft SQL Server, PostgreSQL and MongoDB, respectively (Fig. 14). It can be seen that relational database management systems (RDBMS) have been, and still are, occupying a large proportion of applications.

However, in the aforementioned ranking , NoSQL (Not only SQL) databases are getting more popular with MongoDB (rank 5) and Redis (rank 7). This can be explained by the emergence and rapid development of social media applications such as Facebook, Twitter, Instagram, and Youtube. When using these applications, users have been continuously generating a huge amount of data, such as posts, messages, images, videos, etc. These data not only have large content quantities but also have various heterogeneous structures. On the other hand, RDBMS was not designed for storing and processing this type of data. Therefore, many engineers have turned to NoSQL databases such as MongoDB to deal with these complicated structured data. From there, migrating data from RDBMS to NoSQL have been an on-going trend.

As a result, in this experiment, we used the relational data format as the input and NoSQL as the output. Among the many types of database management systems, we cho se MySQL and MongoDB because they are widely-used (DB-Engines ranks [21]), and the most important reason is that they are open-source and free-to-used.

Experiment Model

The system is presumed to be able to successfully convert data from MySQL to MongoDB if it can satisfy two tasks:

1. Converting and expressing the properties of the data set from the old database in the new database.
2. Converting and migrating data from the old database to the new one. The conversion must ensure the full transfer of all tables and rows of data from MySQL to MongoDB without losing or changing the data itself.

Based on the system architecture of the framework, in this version, we only implemented some of the modules required for converting from RDBMS to NoSQL, unnecessary modules were not implemented. The intermediate data type is JSON, and MongoDB is used to store these data.

All modules are developed using CKAN—an open-source open data portal tool [22]. Details of the modules are as follows (Fig. 15):

Input and Output Gate (IO)

This module used CKAN extension as an interface to interact with users. This module included two components:

- Input gate: This component receives input data, in the form of SQL file, from the user. Using this file, the system will restore the MySQL database for the next steps.
- Output gate: This component returns the conversion results to the user. The results consist of the backup file of the destination database, the external files, and the log file containing the conversion result information.

364 systems in ranking, March 2021

Rank Mar 2021	Rank Feb 2021	Rank Mar 2020	DBMS	Database Model	Score Mar 2021	Score Feb 2021	Score Mar 2020
1.	1.	1.	Oracle	Relational, Multi-model	1321.73	+5.06	-18.91
2.	2.	2.	MySQL	Relational, Multi-model	1254.83	+11.46	-4.90
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1015.30	-7.63	-82.55
4.	4.	4.	PostgreSQL	Relational, Multi-model	549.29	-1.67	+35.37
5.	5.	5.	MongoDB	Document, Multi-model	462.39	+3.44	+24.78
6.	6.	6.	IBM Db2	Relational, Multi-model	156.01	-1.60	-6.55
7.	7.	↑ 8.	Redis	Key-value, Multi-model	154.15	+1.58	+6.57
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	152.34	+1.34	+3.17
9.	9.	↑ 10.	SQLite	Relational	122.64	-0.53	+0.69
10.	↑ 11.	↓ 9.	Microsoft Access	Relational	118.14	+3.97	-7.00

Fig. 14 DB-engines on March 2021 [21]

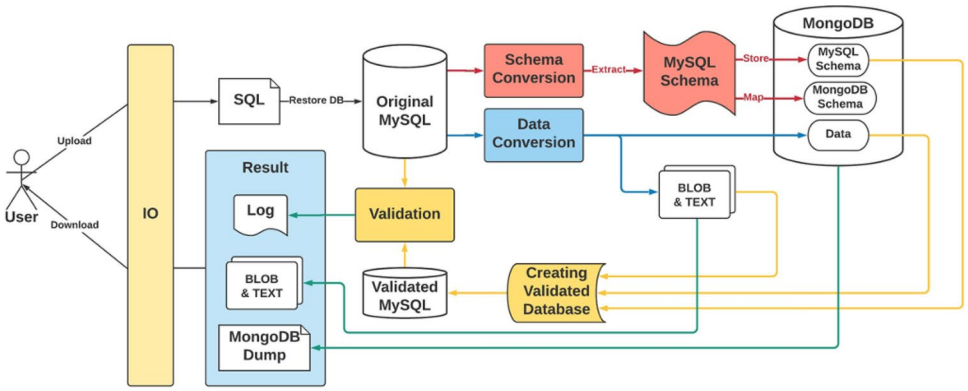


Fig. 15 Workflow of data conversion system

Schema Conversion for RDBMS

This module consists of two main jobs: extracting and converting schema. Details of this module are described in Table 2. The two main jobs of this module could be described as follows:

- **Extracting the MySQL schema** First, we used SchemaCrawler (a database schema discovery and comprehension tool) to extract the schema from the MySQL database. Schema could be generated in many formats, such as plaintext, HTML, JSON, YAML, or even an image file. However, we decided to choose JSON because this data format is suitable for computers to read and process [19]. This schema would be an important object for the next steps: migrating the data from relational model to document-store model.
- **Converting the schema** MongoDB uses jsonSchema to express the structure and properties of the documents in the collections. Schema validator is a tool used to define schema in MongoDB. Schema validator accomplishes that using jsonSchema to define data types and constraints fields in the documents. To put it simply, schema validator is quite similar to data definition language in RDBMS, a language used in RDBMS to define the data structures. Therefore, each schema validator will be applied to each different collection, and perform the check with the task of adding and modifying data in that collection. At the moment, MongoDB jsonSchema cannot fully express all data constraints that are available in MySQL, so some properties in the MySQL schema will be explicitly shown in jsonSchema, the rest will be saved as text.

The reasons for choosing to explicitly implement or just saving schema as text are explained as follows:

- Column data type: This is an important property that needs to be clearly and accurately presented. Because in every context, the data type also represents a semantic part of the data. For example, the numerical values 215 and the string "215" are considered to be two completely different values. So this constraints must be done in *jsonSchema*.
- Primary key, foreign key: Obviously, primary keys and foreign keys must be presented because the relationship between tables represents part of the meaning of a data set through its schema structure. However, MongoDB's data relational implementation techniques are different from MySQL. While the relation of MySQL is based on table technique (equivalent to collection), the relation of MongoDB is based on documents (equivalent to row). For that reason, the relational techniques on MongoDB do not belong to jsonSchema above but will be implemented later.
- Index: Theoretically, although the index is stored with the data in the database, the index itself is not considered to be the data of that data set, but just a technique for processing the data faster. MySQL and MongoDB both have index feature, but because the difference of these two database management systems' data model,

Table 2 Schema conversion method

Column data type	MongoDB schema validator
Primary key, foreign key	Database reference
Index	Index
Constraint: check, not null, default, unique	Not converted, only saved as text
Triggers, functions, and stored procedures	Not converted, only saved as text

applying MySQL index strategy in MongoDB may not be effective. Because of that, we chose not to implement the index. However, all the indexes are stored in the schema so that the user can decide whether or not to apply them in the new database.

- Constraint: the MySQL constraint listed in Table 2 cannot be expressed correctly in MongoDB, so all these constraints are saved as the text for users' reference.
- Triggers, functions, and procedures: Similar to indexes, these techniques themselves do not belong to the database's data, but they are more similar to business tasks at the database level. So these features are saved as text.

Data Conversion for RDBMS

This module has two main jobs: migrating data from MySQL to MongoDB and converting MySQL foreign key to reference in MongoDB.

Data migration In this step, the module proceeds to retrieve all data from the tables of MySQL and saves it into MongoDB. The equivalent terms for the organization of data stored in MySQL and MongoDB for data movement are listed in Table 3. To achieve the goal of data migration, there are two tasks that must be done:

- Creating the appropriate SQL queries to get data from MySQL.
- Converting the data (if necessary) and save them to MongoDB.

Based on the above schema, this module can automatically generate queries to retrieve data from MySQL. Once the data are retrieved, the next issue is to convert the data type (if necessary) to conform to MongoDB. When using Python for the system implementation, by reading MySQL data into a Python variable, MySQL data type will be converted to equivalent Python data type. But in real scenario , some data types cannot be immediately stored in MongoDB's database and will raise errors. Therefore, it is necessary to check the data types of MySQL for proper type conversion.

As an exception, values of the BLOB and TEXT data types of MySQL cannot be stored directly in a BSON document due to size limitations. So these data values will be written to external files, and also saved the reference in the BSON document.

Table 3 Data migration MySQL to MongoDB-equivalent terms

MySQL	MongoDB
Database	Database
Table	Collection
Row	Document
Column	Field

Converting foreign key to reference In MySQL, the relationships between tables are very crucial. Those relationships partly express the meaning of the data set through its structure. Besides that, the relationship ensures that MySQL maintains consistency, integrity and avoids data redundancy through normalization. Therefore, the conversion of these relationships in MongoDB is required and cannot be ignored. MongoDB has two techniques to express the relationship between data records: the referencing approach and the embedding approach. We have analyzed and found the suitable technique for the implementation as follows:

- **The referencing approach** First, the referencing approach uses a database reference, also known as a document reference in MongoDB, to demonstrate the relationships between documents. This approach is similar to the one in MySQL, using an object that acts as a foreign key. This object has three pieces of information including the document's primary key (the ObjectId id), the collection name, and the database name it needs to refer to. Hence, this type of document reference in MongoDB fully demonstrates the properties of foreign key constraints and foreign key columns in MySQL. Also, because the approach and implementation are somewhat similar, this referencing does not take full advantage of the advantages that MongoDB offers. For example, when you need to query data, you still have to execute the lookup function (equivalent to a JOIN in MySQL) to get all the necessary data.
- **The embedding approach** In contrast to the referencing approach, in the embedding approach, the referenced document will be included in the reference document, thereby creating the concept of "rich document" in MongoDB. Then every time data are required, the entire (rich) document will be used immediately, instead of having to do JOIN operations like in MySQL or lookup in MongoDB to get all the data you need. This approach makes searching on MongoDB significantly faster than MySQL.

The benefits of using embedding approach are quite obvious, but the risks in return are also worth some consideration, especially when using and applying these techniques inappropriately. First, the embedding approach could easily cause data redundancy. When using embedding method, an original document could be embedded in a lot of other documents, thus created data redundancy. When the need to update the embedded document arises, if the updating process is not handled well, it is easy to create data inconsistency: the embedded data in some documents are updated while some other embedded data in other documents are not. Second, in the first example, we have only examined

the scenario in which there existed only one original document that is need to be embedded. However, in common scenarios, a database will use foreign keys much more. The embedding approach will then create a massive amount of embedded documents. The creation of this many embedded documents will then lead to the massive update of documents; thus, the performance will be hindered. Thereby, we can see that the embedding approach is highly dangerous if the database is not well designed or misused. Because of that, it is recommended to use both in conjunction with each other. The problem of deciding which document to embed is complex and should then be considered carefully by the database designer.

From the analysis of these two approaches, we have drawn the following comment. The advantages of the embedding approach is that this approach makes a good use of MongoDB's advantages, making the performance system much more better, but only if the designer is good enough. But when misused or not well designed, it will encumber the system performance. The reference approach's advantage is that it will never cause any data redundancy or data inconsistency at the cost of the system performance. But the cost of system performance is only noticeable in some cases when retrieving data.

In the context that our data conversion system is fully automated, applying the embedding approach to a relational implementation in MySQL can introduce a lot of risks into our system. Therefore, in this version, we decided to use document reference for relational transformation. After the conversion is complete, users can completely embed documents according to their preferences, based on the available references generated by the system (Fig. 16).

Validation

The experiment version has only output validation, which is responsible for ensuring that the data before and after conversion is still the same in terms of data value, data type, and the relationships between records. To do this, the module converts the intermediate MongoDB database into another MySQL database, then compares these two MySQL databases to check the conversion result. The two aforementioned databases is compared on the basis of the structure of databases, the number of records in the databases, and the records in the databases.

Evaluation

The test was conducted on 9 different databases, varied in structure and size of data. The detail about these databases (the number of tables, the total number of attributes or columns and the total number of records or rows in each database) is presented in Table 4.

The results showed that all 9 databases were successfully converted, ensuring the values in the data records were not changed, the number of records in the database is correct and the relationship between the records was still fully expressed. Thereby this experiment version proved the feasibility of the framework when it will be developed for practical application.

Table 4 The detail about the databases that are used in the experiment

Name of input database	Number of tables	Number of columns	Number of rows
Chinook	11	64	15.587
Classicmodels	8	59	3864
Employees	6	31	3.911.594
Northwind	8	78	3.202
Sakila	16	132	47.271
Sample-ip	2	4	5.641.408
Sample-staff	16	170	6.148.999
World	3	24	5.269
Xtoss	110	994	79.355

Fig. 16 Referencing vs embedding

Conclusions

Data conversion is an emerging topic closely related to various fields, including open data, Internet of Things. Recent studies have shown the important role and usefulness of data conversion task in data integration systems. However, these current data conversion tools are still relatively simple, and does not fully consider the diversity of input data sources and data formats or the challenging context of big data. In particular, there are only a few researches that have the ability to expand to deal with various data formats or completely new data formats. The lack of the aforementioned researches was our inspiration. In this paper, we proposed a data transformation framework that allows users to declare the characteristic of new data and enables to convert the data into the desired formats. The novelty of our framework is the openness of our framework: users can add or change the data formats in our framework.

To evaluate the model, we did an experiment by building a system with CKAN, MySQL and MongoDB. Using JSON as the intermediate data type, the system performed our proposed framework very smoothly and could be considered as a use case for MySQL and MongoDB.

However, there were still some related challenges such as schema conversion, schema mapping, efficient data conversion, distributed data conversion techniques, ontology mapping between schema, incremental data conversion, remote data conversion, etc. Other than aforementioned problems, there are also even more challenges in how to use the data: how to protect the privacy of the users who provide the data, how to prove the provenance of the data or how to use the data in a system, etc. All these problems can be considered as interesting research topics for us to gradually solve in future.

Acknowledgements This work is supported by a project with the Department of Science and Technology, Ho Chi Minh City, Vietnam (contract with HCMUT no. 42/2019/HD-QPTKHCN, dated 11/7/2019).

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Sing LC, Youwei J, Zhekang D, Dongxiao W, Yingshan T, Hong LQ, Wong Richard TK, Zobaa Ahmed F, Ruiheng W, Lei LL. A review of technical standards for smart cities. Clean Technol. 2020;20:20.
2. McLaren D, Agyeman J. Sharing cities: a case for truly smart and sustainable cities. New York: MIT Press; 2015.
3. Payam B, Maria B-E, Ralf T. Challenges for quality of data in smart cities. ACM J Data Inf Qual. 2015;20:20.
4. Federal Highway Administration, U.S. Department of Transportation. Data Integration Primer. https://www.fhwa.dot.gov/asset/dataintegration/if10019/dip00.cfm. (2010).
5. Nguyen LH, Le HT, Dang TK. A comparative study of the some methods used in constructing coresets for clustering large datasets. SN Comput Sci. 2020;1:4 (Online ISSN: 2661-8907).
6. Dang TK, Nguyen QP, Nguyen VS. Evaluating session-based recommendation approaches on datasets from different domains. In: Proceedings of the 6th future data and security engineering, LNCS 11814. Springer; 2019. p. 577–592.
7. Dang TK, Huy TM, Hoang NL. An elastic data conversion framework for data integration system. In: International conference on future data and security engineering FDSE 2020; 2020.
8. Ermilov I, Stadler C, Auer S. CSV2RDF: user-driven CSV to RDF mass conversion framework. In: Proceedings of the 9th international conference on semantic systems; 2013.
9. Rocha L, et al. A framework for migrating relational datasets to NoSQL1. Proced Comput Sci. 2015;51:2593–602.
10. Hyeonjeong L, Hoseok J, Miyoung S, Ohseok K. Developing a semi-automatic data conversion tool for Korean ecological data standardization. J Ecol Environ. 2017;41:11.
11. Vathoopan M, Brandenbourger B, George A, Zoitl A. Towards an integrated plant engineering process using a data conversion tool for AutomationML. In: IEEE international conference on industrial technology; 2017. p. 1205–10.
12. Paiva L, Pereira P, Almeida B, Maló P, Hyvärinen J, Klobut K, Dimitriou V, Hassan T. Interoperability: a data conversion framework to support energy simulation. Proceedings. 2017;1(7):695 ISSN: 2504-3900.
13. Dong XL, Srivastava D. Big data integration. San Rafael: Morgan & Claypool Publishers; 2015. p. 198.
14. Obitko M, Jirkovský V. Big data semantics in industry 4.0. In: Industrial applications of holonic and multi-agent systems. Lecture notes in computer science, vol 9266; 2015. p. 217–29.
15. Knoblock CA, Szekely P. Exploiting semantics for big data integration. AI Mag. 2015;36(1):25–38.
16. Information Builders. Real world strategies for big data—tackling the most common challenges with big data integration—a white paper; 2016.
17. Microsoft. SQL server integration services. https://docs.microsoft.com/en-us/sql/integration-services/sql-server-integration-services (2017).
18. Talend (2017). Talend data integration. https://www.talend.com/.
19. Dang TK, Huy TM, Hoang NL. Intermediate data format for the elastic data conversion framework. In: International conference on ubiquitous information management and communication IMCOM 2021; 2021.
20. Stack overflow insight. https://insights.stackoverflow.com/.
21. DB-Engines. DB-engines ranking. https://db-engines.com/en/ranking.
22. CKAN—The open source data portal software. https://ckan.org.