Improved method of selecting data in a nonrelational database

Roman Ceresnak
Faculty of Management Science and Informatics
University of Zilina
Zilina, Slovakia
roman.ceresnak@fri.uniza.sk

Michal Kvet
Faculty of Management Science and Informatics
University of Zilina
Zilina, Slovakia
michal.kvet@fri.uniza.sk

Karol Matiasko
Faculty of Management Science and Informatics
University of Zilina
Zilina, Slovakia
karol.matiasko@fri.uniza.sk

***Abstract*— People surround themselves with data in many ways, which evokes the need for correct ways of storing the data. Nowadays, the trend tends to lean in favor of data storing in nonrelational (or NoSQL) databases. These databases are used in various user applications, which need a huge volume of the data highly accessible and do not require big data consistency. The problem of the data growth and its storing in the nonrelational databases results in the decreasing efficiency of searching in data. In the paper, we present use of very popular in-memory database in order to help us with this lack of efficiency of the data searching. This paper examines the data searching in applications hosted by Amazon cloud service while using nonrelational database DynamoDB. We develop new procedures to provide faster response to user and to obtain the data using nonrelational database DynamoDB. These procedures provide the queried data and subsequently, transfer them into the memory. The given procedures are based on two methods. The first method is a recognition of values, the user refers to and the provision of this data to the in-memory database. The second method is related to the automatic storing of the data transferred to the in-memory database. We perform number of experiments, which are describing a limitation of efficiency/inefficiency from a perspective computational time.**

Keywords—Data Searching, NoSQL databases, in-memory databases, DynamoDB

I. Introduction

The growth of population and amount of produced data caused, that many conventionally used procedures and systems, like traditional databases, started gradually losing their efficiency. In contrast with relational databases [2], NoSQL databases process and manage the big data, characterized by 3V (volume, variety, velocity) [3]. NoSQL databases are critical in support of various applications, which need various levels of performance consistency, availability, and scalability [4]. Social media such as Twitter and Facebook [5] generate exabytes of the data daily, which exceed processing capabilities of relational databases. These applications demand high performance, but they do not have to demand strong consistency.

It is impossible to achieve the same efficiency of searching in a large amount of the data while working with the nonrelational and relational databases. Regarding the searching in the nonrelational databases, the data, which do not have to meet the strict structural demands of system (RDMBS), are stored, because the data for the searching can be texted, semi-structured or unstructured. A search engine database is created to help the users in fast finding of the information they need in a highly qualified and cost-effective way. These databases are optimized for the use of keywords and usually offer specialized methods such as full-text searching or searching with the use of complicated expressions of various types.

Search engine database consists of two main parts. The first part is adding a search engine database index to the data. When the user queries for data, relevant results are quickly returned with the help of the search engine database index. This fast and responsive way of data searching is possible since instead of direct searching for queried text, these databases search for relevant index in the database. This can be thought of as an equivalent to looking for page number related to the term in book index, in contrast to searching for individual words on every page in the book. This type of index is called an inverted index because it transfers the data structure-oriented on a page to the data structure oriented at the keywords.

Second part of the search engine database is the data searching can be made more efficient with the use of in-memory databases. The in-memory database (IMDB) is a computer system, which stores and searches the data records situated in the main memory of computer e.g., in RAM memory. IDMB is an advantageous approach to the data storing since traditional databases work with a data access delay as a result of storing data on medias with higher time of access such as hard discs, SSDs and so on. This means, that IDMB is useful when fast reading and recording of data is crucial.

Most of IMDB implementation preserve data in the RAM. Some implementations use IDMB with the combination of disk part of the system, but RAM is still primary storing medium. Some IDMBs also store the data on disk as a preventive measure to minimalize the risk of the data loss, since RAM is volatile e.g., the data is lost when a computer loses electric energy.

The majority of IDMB also prevents the data loss in chosen data center (property known as “high availability”) preserving the copies (technically called replicas) of all the data records on several computers in a cluster. This data redundancy secures, that when any kind of error makes whichever computer in the network not available, no data record could be lost. Among the most popular in-memory databases, which use query languages for data searching, belong databases such as Redis, Memcached and similar products. Artificial intelligence can be used in order to help us with the purpose of transfer of the data situated in the nonrelational database DynamoDB.

The problem related to the data transfer from the conventional disk-based database to the in-memory database is known as velocity of the data searching and the transfer efficiency. Artificial intelligence was used for these purposes, which secures this transfer and so it also makes the data

978-1-6654-3692-2/21/$31.00 ©2021 IEEE

searching more effective. The main objectives of this paper are as follow:

- We create a new procedure for processing the data in the memory,
- We reduce the time needed for the record searching in the nonrelational databases,
- We define the methods of automatic adjustment of the data growth to the size of in-memory database.

The rest of the paper is structured as follows. "State of the art" section examines the work related to objective of presented paper. Part III presents our proposed solution - designed searching model and the characteristics of this model. "The data transfer" and "The index creation" parts describe the performance of the operation in DynamoDB. "Experiments" part describes performed experimental testing and subsequently their results.

II. State of the art

A comparison between relational data models and nonrelational (NoSQL) data models was already stated in various papers. For example, comparisons between these two types of database were focused on the times needed to perform basic database operations such as data selection, data insert, data update, and deletion of data [1].

Several statistics point out the fact the most common operation, which is demanded in the relational and nonrelational database, is data selection operation [1, 2]. Number of authors presented in their papers [3], that the time needed to get the data in the nonrelational database is significantly higher as the time needed to get the same data in the relational database. Computational time of operations is crucial not only in database systems, but all problems related to the computer science [4, 5], hence need for optimization of conventional approaches. As a way to accelerate or improve the time needed to get the data from the nonrelational database, it is possible to store the data to buffer memory and by this reduce repeated searching in the nonrelational database. With the use of this method not only computational time is reduced, but also number of accesses to the database is lowered [6].

The authors performed various comparisons between in-memory databases such as Redis, Memcached, and nonrelational databases Mongo, Casandra, and H2 [7]. One of the main findings of these works is the verification of data update and deleting of the data with its increasing amount. During our research related to the topic of the paper, we noticed, that papers focus on problem-solving in the context of increasing amount of the data.

In the authors created a module by using the library Lontar, which sends the data to the relational database with the use of Hibernate as a framework and a relational mapper, in the case of user's demand. Subsequently, Hibernate accesses the MySQL database and maps the relational data to object-oriented [8], and then it sends the data to the nonrelational database. The searching then works with the help of the mapper, so Lontar is able to read the data in a relationship. According to the authors, searching for data in chosen data files resulted in better times for nonrelational database MongoDB than for relational database MySQL. However, in certain situations, the relational database got better results than the nonrelational database.

The authors of [9] introduced a framework capable of data manipulation in order to overcome the problems related to the decreasing efficiency of the searching in the nonrelational databases. Before performing the basic operations of data selection, data insertion, data update, and data delete this framework uses mapper function. The main role of the mapper is to change the data on the base of rules, to such form, which complies with principles of nonrelational database MongoDB more. With the use of this module, the speed of data searching is in majority of cases higher in nonrelational database MongoDB compared to the relational database MySQL. Another concept used in this framework is the Cataloging module, which uses JSP (JAVA) as a web programming technology and MySQL as DBMS. There are two frameworks supporting this concept, Structs and Hibernate. Structs are used to set users' interface and the Hibernate regime is used to map the relational data to the object-oriented data, which will be used by JSP.

In our work, we also design and implement framework, which uses two database types. The first database is nonrelational database DynamoDB serving as a primary data storage. In the case, when user demands data, the values will not be directly given to them from the nonrelational database, but the values will be transferred to the in-memory database. The main challenge in this approach is to transfer the data from the nonrelational model to the in-memory model.

III. Our Contribution

In this part of presented paper, we introduce two modules which are used in order to make the searching in the nonrelational databases more effective. Data Cached Module (DCM) and Data Elastic Module (DEM). DCM serves as a data storehouse, whose role is the data transfer to the buffer memory. DEM serves as a tool for automatic adjusting to the data size.

A. Data Cached Module

The created module serves as a way of the data processing in the memory. We connected the data we store in the nonrelational database DynamoDB to highly available buffer memory Amazon DynamoDB Accelerator, well known in short as DAX, with the help of API interface. This method helps us with Side-cache access, Read-through cache access and Write-through cache access as follows:

a) Side-cache – This principle helps us with high overload during the reading of information from the memory. The principle works as follow (see Fig. 1):
   1. An application first tries to load the data for a given key-value couple from the buffer memory. If the buffer contains queried data, the value of this key-value pair is returned as and output of the operation. Otherwise, step 2 follows.
   2. Since the demanded key-value pair was not found in the buffer, the application loads the data from the basic data storage.
   3. A key-value pair from step 2 is written to the buffer memory to make sure the data are present when the application needs to load the data again in the future uses of similar query.

978-1-6654-3692-2/21/$31.00 ©2021 IEEE

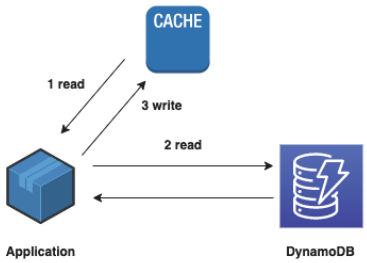

Figure 1. Side-cache algorithm

b) Read-through cache – DAX is a buffer memory for the reading - it is compatible with API for the reading of DynamoDB and stores the results *GetItem, BatchGetlem, Scan*, and *Query* to the buffer memory, if they are not currently in DAX. The buffer memory for the reading is effective in high working loads. This principle works as follow (see Fig. 2):
   1. Regarding a key-value couple from the application, algorithm first tries to load the data from DAX. In the case the buffer contains queried data, the value of this key-value pair is returned as and output of the operation. Otherwise, step 2 follows.
   2. Transparently for the application, if a semi-memory happens, DAX will load a couple of key-value from DynamoDB.
   3. To make the data available for every reading that follows, a key-value pair is stored in the semi-memory of DAX.
   4. A key-value couple is then returned to the application.

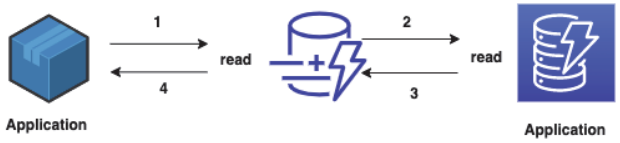

Figure 2. Read-through cache algorithm

c) Write-through cache – Similarly to the semi-memory for the reading, a semi-memory for the data writing also operates in a line with the database and updates the semi-memory, when the data is written to the basic data storage. DAX has also buffered memory for writing since it stores to the buffer memory (or updates) the items with *PutItem, UpdateItem, DeleteItem* and *BatchWriteItem* API, because the data is written or updated in DynamoDB. At first, DAX is updated (everything is transparent for the application). The following steps indicate a procedure for buffer memory of write-through type (see Fig. 3):
   1. The application will write itself to endpoint DAX for a given key-value couple.
   2. DAX will catch the writing and then will write a key-value pair into the DynamoDB.
   3. After the successful writing, DAX hydrates buffer memory with a new value so whichever following the reading of the same couple key-value results in a finding of the buffer memory. If the writing is unsuccessful an exception will return to the application.
   4. Confirmation of successful writing will then return to the application.

Figure 3. Write-through cache

B. Data Elastic Module

The created module works with the data which increases demands for the storage space. Nonrelational database DynamoDB fulfills the task of a wide data storage and in the case of the data transfer to the in-memory database, size of the data can grow from several megabytes to several gigabytes very easily. This problem is solved with the use of the created module.

We configured monitoring of metrics in cloud service Amazon with the help of Amazon CloudWatch service. The mentioned service makes it possible to edit (to add and to remove) new computation units in the case of enabling of horizontal or vertical scaling. An advantageous characteristic of this method is the horizontal scaling which, in the case of a large number of the data uploads, invokes warning of system overload and a script for the reading of information from other replicas in service CloudWatch. The horizontal replica is the part of the script performed automatically during the configuration of the in-memory database DAX with the following script:

```
aws dax decrease-replication-factor \
  --cluster-name MyNewCluster  \
  --new-replication-factor 3
```

The monitoring of the metrics in the same way with our method also makes the vertical scaling possible - the scaling by addition or removal of the computation units (Fig. 4).

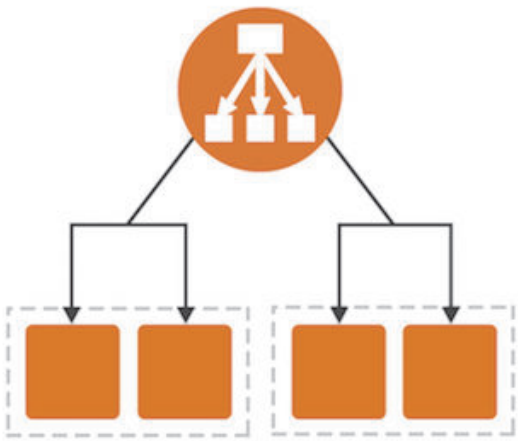

Figure 4. Application Load Balancer for in-memory database

In a situation, when the values retrieved from the nonrelational database do not fit the in-memory database, the event is invoked again with the help of service CloudWatch, which will cause the addition of a new computation unit. When the computation unit is not needed anymore, an instance is automatically released, which results in saving of the buffer memory and optimization of a price.

IV. EXPERIMETNS

In the very first step, we created a simple database model presented in the Fig 5. This database model consists of two tables - *user* and *comment*. These two tables are interconnected by identification relationship of type *1:n*, which means one user can create number of various comments and various comments in the table belong to one user.

Subsequently, we compared various commands, whose aim is to retrieve information from implemented database model. The objective is to measure time of computation of simple queries in conventional systems.

In the sections A and B, we present measurements for four sizes of queries in the relational database Oracle (section A) and nonrelational database DynamoDB (section B).

Since an important aspect of this paper is use of the in-memory database, we also compare the times of various operations during data selection in section C.

In the section D, we compare these conventional approaches to the problem with proposed solution described in the secion III.

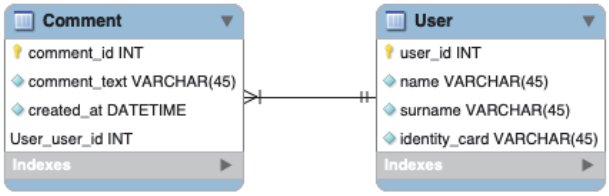

Figure 5. Database model

A. Experiments on relational database Oracle

We inserted 1000 records into table *user* and 1000 records into table *comments* which are based on the defined structure. In the case of this paper, we are interested only in information about the time needed for record searching. We created 3 data selection commands for these purposes:

```
(1) SELECT name, surname FROM user
    JOIN comment USING (user_id);

(2) SELECT * FROM user
    JOIN comment USING (user_id)
    WHERE comment_text LIKE "%today%";

(3) SELECT  name, surname,
    to_char(created_at, 'YYYY-MM-DD HH24:MI:SS') ca
    FROM user
    JOIN comment USING (user_id)
    WHERE ca >= TRUNC(current_date)
    and ca < TRUNC (current_date) + 1;
```

The first 1000 records served as benchmarking records to us - from these results, we continued our research. The main purpose of the nonrelational databases is to effectively store large amounts of data, and that is, why the records for other purposes will be created with the sizes of 100 000 records for table *user* and 100 000 records for the table *comment*. Subsequently, all records are deleted, and the dataset of size 10 000 000 records will be inserted to both tables. As the last size of the datasets, we chose a value of 100 000 000 records for *user* and *comment* tables.

A generator was used for record creation with the size of 1 000, 100 000, 10 000 000 and 100 000 000, which can be found on the following address:

https://www.generatedata.com/

The generator provides an option to define names and types of attributes and to generate an arbitrary number of the values. After fulfilling the tables by the generated values, we recorder the times needed to perform operations (1), (2), and (3). These measured times are presented in the Table 1. All measured values for processing of commands (1), (2), and (3) are stated in seconds.

Table 1. Measure time for operations (1), (2) and (3)

Count of records/operation	1 000	100 000	10 000 000	100 000 000
(1)	0,0020	0,004	0,028	0,44
(2)	0,0021	0,0042	0,031	0,45
(3)	0,0021	0,0044	0,030	0,45

B. Experiments on nonrelational database DynamoDB

We used commands (1), (2), and (3) to find out the velocity of data queries in the nonrelational database DynamoDB. The values inserted into the database were left the same as in experiments in the relational database. The structure is fully the same as is presented in the Fig. 3.

Table 2. Measure time for operations (1), (2) and (3) in DynamoDB

Count of records/operation	1 000	100 000	10 000 000	100 000 000
(1)	0,0035	0,0064	0,047	0,82
(2)	0,0035	0,0067	0,048	0,83
(3)	0,0037	0,0068	0,046	0,82

The values, needed to get the data from the nonrelational database DynamoDB are presented in the Table 2. All measured values for commands (1), (2), and (3) are stated in seconds.

When comparing values of computation time of the same operations between the relational and nonrelational databases, we can clearly see the significant difference. We

978-1-6654-3692-2/21/$31.00 ©2021 IEEE

can conclude that data selection operation in nonrelational databases is less effective than in the relational database Oracle.

C. Experiments for the in-memory database Redis

The data storing in the in-memory database is diametrically different than in the relational or nonrelational databases. Except for the mentioned fact, there is also problem with the amount of data caused by computer memory limitations. The computer memory used for testing purposes was about 8 GB.

Table 3. Structure of the data in the in-memory database

ID	Name	Surname	Identity_card
1	John	Harper	12341324
2	Joe	Bush	12341234
3	George	Obama	23524675
.....	.....	.....	.....
.....	.....	....	.....
1000	Alan	Felps	45674866

As seen in the Table 3, we created records with the identical structure to the previously used table *user* - *ID, Name, Surname* and *Identity_card*. Specifically, we create datasets of 100, 300, 500 and 1000 records.

Three commands were created for the purposes of the testing of the in-memory databases' effectiveness. These commands were structured as follows:

(4) MGET Name

(5) MGET Name, Surname

(6) MGET Name, Surname, Identity_card

(7) MGET Name, Surname, Identity_card, Age

We applied the same principle during filling the database as in previous steps. In the specific case, we inserted the generated values to the database, tested operations (4), (5), (6), and (7), and recorded the measured values. Subsequently, we deleted all the records and inserted the next dataset of 300 records to the database. We continued in this fashion until the size of 1000 records in the database was reached. The measured values for the operations are presented in the Table 4.

Table 4. Measure time for Redis database

Count of records/operation	100	300	500	1 000
(4)	0,00020	0,00022	0,00021	0,00028
(5)	0,00021	0,00023	0,00025	0,00029
(6)	0,00021	0,00022	0,00025	0,00028
(7)	0,00023	0,00024	0,00028	0,00032

All measured results in the Table 4 are presented in seconds. The seventh operation is influenced by the fact, that value "age" does not exist. As can be seen, the measured values are not diametrically different with the increasing number of records. It is necessary to point out, that with defined growth of the records, it is the logic fact mirroring the efficiency of the searching in the memory.

D. Comparison of conventional methods and proposed method

The values we measured in the experimental activity presented in the sections A, B, and C serve for comparison of conventional methods with the proposed method. The compared values operation (1) on the datasets of 1 000 and 1 000 000 records are presented in the Table 5.

Table 5. Comparison of query performance

Count of records/operation	1 000	1 000 000
Oracle	0,0020	0,44
DynamoDB	0,0035	0,82
Our Approach	0,0033	0,42

As seen in the Table 5, the values measured while using operation (1) with a low number of records in the table, do not hint towards any big improvement of the searching in the nonrelational table. This is influenced by the data transfer to the memory. A factor of the transfer indicates a necessity to transfer the data from nonrelational database DynamoDB to buffer memory DynamoDAX, which takes a certain time which is combined with the computational time of the query processing itself. This means, that in the operation of data selection, the data are physically retrieved from in-memory database, not from the nonrelational database DynamoDB.

Based on the data transfer, it was possible to also compare the measured times of the experiments with the in-memory database and the data transferred to DynamoDAX with the size of 100 and 500 records and with the use of operation (5).

Table 6. In memory query performance

Count of records/operation	100	500
Redis	0,00020	0,00021
DynamoDAX	0,00015	0,00017

The values recorded in the Table 6 show efficiency of the data transfer. As cane be seen, the values in buffer memory Dynamo DAX are more effective from the time perspective than in-memory database Redis.

Whole achieved results related to operation data selection in nonrelational database DynamoDB were not, before the application of our method, timely the same effect than after the application of our method. With the use of machine learning and transferring the data to the database in memory, the efficiency of operation data selection in the nonrelational

database became more effective after achieving 1000 000 records than with the searching of the data in relational database Oracle. A huge advantage, that results in using of cloud storage Amazon, is related also to the possibility of automatic scaling respectively adding of performance and increasing of the storage not only in nonrelational database DynamoDB, but mostly in the database in memory alternatively, if we do not need as many calculation units, so the reduction of the size of the data storage happens, and so the decreasing of the cost related to running of our designed method happens.

Conclusion

NoSQL databases play a significant role in storing and processing large amounts of data and they are used in various wider social applications such as Twitter, Facebook, Google, and Yahoo, but they help also with the decision support or in the advanced analyses. These databases became the master of high effectiveness of storing and availability of the large datasets. With this came the loss of the effective searching methods, which can be found in the traditional databases. This paper was focused on the question of the data searching time optimization in NoSQL databases, specifically DynamoDB in the cloud environment of Amazon.

In this paper, we developed the data searching algorithm, which can make the searching in a nonrelational database DynamoDB more effective. The designed algorithm is composed of two parts, the first part is based on the principle of caching the data from the nonrelational database DynamoDB to buffer memory DynamoDAX. The second part is based on the effective data and system management - in the case of the large amount of data, system automatically increases the number of computation units of the buffer memory, and by this adjusts the size of the database to the increasing needs of the size of incoming data. This fact relieved us from the limitation of the database size towards the data.

The experiments provided us with useful information about the performance and the effectiveness of the created method. It is noted, that the system for processing artificial intelligence demanded higher overhead costs together with automatic creation of the database in memory, but this system was able to make the process of searching in the nonrelational database more effective. On the basis of the experiments, it is clearly seen, the created method is more and more effective with the increasing data amount, which is done by the data transfer to the memory.

Our future work will focus on a generalization of this model and provision of user interface for full use of the created procedure not only for Amazon cloud but also for other in-memory databases such as Redis and Memcached. We also plan to evaluate the suggested systems empirically, from a perspective of consistency and performance in other environments, which need fast response for the data demand.

Acknowledgment

This publication was realized with support of Operational Program Integrated Infrastructure 2014 - 2020 of the project: Intelligent operating and processing systems for UAVs, code ITMS 313011V422, co-financed by the European Regional Development Fund.

EUROPEAN UNION
European Regional Development Fund
OP Integrated Infrastructure 2014 – 2020

MINISTRY
OF TRANSPORT
AND CONSTRUCTION
OF THE SLOVAK REPUBLIC

References

[1] R. Čerešňák and M. Kvet, “Comparison of query performance in relational a non-relation databases,” *Transp. Res. Procedia*, 2019.

[2] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, 2013, pp. 15-19, doi: 10.1109/PACRIM.2013.6625441.

[3] T. N. Khasawneh, M. H. AL-Sahlee and A. A. Safia, "SQL, NewSQL, and NOSQL Databases: A Comparative Survey," *2020 11th International Conference on Information and Communication Systems (ICICS)*, 2020, pp. 013-021, doi: 10.1109/ICICS49469.2020.239513.

[4] Dudáš A., Škrinárová J., Vesel E.: Optimization design for parallel coloring of a set of graphs in the High-Performance Computing. In: Proceedings of 2019 IEEE 15th International Scientific Conference on Informatics. pp 93-99. ISBN 978-1-7281-3178-8

[5] Dudáš A., Škrinárová J.: Edge Coloring of Set of Graphs with The Use of Data Decomposition and Clustering. In: IPSI Transactions on internet research : multi-, inter-, and trans-disciplinary issues in computer science and engineering. Vol. 16, no. 2 (2020), pp. 67-74, ISSN 1820-4503

[6] P. T. Hulina and A. R. Hurson, "Reducing average access time of a parallel memory in a database environment by data permutation," *Twenty-Third Annual Hawaii International Conference on System Sciences*, 1990, pp. 65-71 vol.1, doi: 10.1109/HICSS.1990.205101.

[7] I. Pelle, J. Czentye, J. Dóka and B. Sonkoly, "Towards Latency Sensitive Cloud Native Applications: A Performance Study on AWS," *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, 2019, pp. 272-280, doi: 10.1109/CLOUD.2019.00054.

[8] H. Wang, Q. Zhu, J. Shen and S. Cao, "Web-Service-Based Design for Rural Industry by the Local e-Government," *2010 International Conference on Multimedia Information Networking and Security*, 2010, pp. 230-235, doi: 10.1109/MINES.2010.58.

[9] G. Karnitis and G. Arnicans, "Migration of Relational Database to Document-Oriented Database: Structure Denormalization and Data Transformation," *2015 7th International Conference on Computational Intelligence, Communication Systems and Networks*, 2015, pp. 113-118, doi: 10.1109/CICSyN.2015.30.

978-1-6654-3692-2/21/$31.00 ©2021 IEEE