Computers, Materials & Continua
DOI: 10.32604/cmc.2022.028326
Article

Tech Science Press

Handling Big Data in Relational Database Management Systems

Kamal ElDahshan[1], Eman Selim[2], Ahmed Ismail Ebada[2], Mohamed Abouhawwash[3,4], Yunyoung Nam[5,*] and Gamal Behery[2]

[1]Faculty of Science, Al-Azhar University, Cairo, Egypt
[2]Faculty of Computer and Artificial Intelligence, Damietta University, Egypt
[3]Department of Mathematics, Faculty of Science, Mansoura University, Mansoura, 35516, Egypt
[4]Department of Computational Mathematics, Science, and Engineering (CMSE), Michigan State University, East Lansing, MI, 48824, USA
[5]Department of Computer Science and Engineering, Soonchunhyang University, Asan, 31538, Korea
*Corresponding Author: Yunyoung Nam. Email: ynam@sch.ac.kr
Received: 07 February 2022; Accepted: 16 March 2022

Abstract: Currently, relational database management systems (RDBMSs) face different challenges in application development due to the massive growth of unstructured and semi-structured data. This introduced new DBMS categories, known as not only structured query language (NoSQL) DBMSs, which do not adhere to the relational model. The migration from relational databases to NoSQL databases is challenging due to the data complexity. This study aims to enhance the storage performance of RDBMSs in handling a variety of data. The paper presents two approaches. The first approach proposes a convenient representation of unstructured data storage. Several extensive experiments were implemented to assess the efficiency of this approach that could result in substantial improvements in the RDBMSs storage. The second approach proposes using the JavaScript Object Notation (JSON) format to represent multivalued attributes and many to many (M:N) relationships in relational databases to create a flexible schema and store semi-structured data. The results indicate that the proposed approaches outperform similar approaches and improve data storage performance, which helps preserve software stability in huge organizations by improving existing software packages whose replacement may be highly costly.

Keywords: Big data; RDBMS; NoSQL DBMSs; MongoDB; MySQL; unstructured data; semi-structured data

1 Introduction

The amount of unstructured and semi-structured data has increased rapidly. Managing such a variety of data is one of the most challenging tasks in DBMSs. In order to effectively manage these data, NoSQL DBMSs play an important role by providing an efficient way of data storage and high levels of availability and scalability. A data transformation from an RDBMS to a NoSQL

DBMS is a challenge because it is based on several factors, such as storage structures, mapping styles, and query structures, and there is no existing standard on NoSQL DBMSs. Furthermore, NoSQL DBMSs have attracted many research interests which showed that they lack some capabilities of RDBMSs. They relax one or more transaction properties provided by RDBMSs such as atomicity, consistency, isolation, and durability (ACID). As a result, many developers have already encountered the shortcomings of the new data repositories in practice, and some have chosen to return to RDBMSs. Nevertheless, RDBMSs face deficiencies in handling data varieties, slow storage and retrieval, scalability difficulties, and low query efficiency when handling big data. This paper proposes two approaches for the following objectives:

- Ameliorate RDBMSs and overcome storage limitations.
- Handling a variety of data "which is considered the main feature of big data".
- Incorporate NoSQL DBMS capabilities in RDBMSs such as scalability and availability.

The rest of the paper is organized as follows. Section 2 introduces big data overview and data storage techniques. Section 3 describes the details of the proposed approaches and the empirical validations of each proposed approach, followed by results and discussion. Finally, in Section 4, the conclusion and further work are presented.

2 Big Data Overview

Big data is an outstanding term that features data availability in all three formats: structured, unstructured and semi-structured formats [1]. This paper discusses a detailed study of big data types and their storage techniques.

2.1 Big Data Types

2.1.1 Structured Data

Since the beginning of the database revolution network, hierarchical, relational, and object-relational databases, the data models deal with structured data [2]. Structured data is obligated to a predefined data model, so these data are easily entered, stored, and analyzed. Most companies like Google use structured data to search the web to understand page content, which is a basic way to describe the web pages.

2.1.2 Unstructured Data

Unstructured data is data that may be in different forms. It is categorized into textual and nontextual data. Textual data types include text documents, personal blogs, web pages, and discussion forums, whereas nontextual data types include sounds, images, and videos. Every day, unstructured data is generated due to the proliferation of smartphones, surveillance devices, and social media. This has created a need for an efficient way to manipulate unstructured data. Unstructured data is managed through one of the following techniques: distributed file systems or NoSQL DBMSs [3,4].

The Hadoop Distributed File System (HDFS) distributes data across multiple clusters. These clusters perform parallel data processing on chunks of data [5]. HDFS maps and reduces unstructured data using Hadoop map-reduce, and then integrates the processing to provide the final results. The processing of unstructured data in HDFS makes it difficult to represent the relationships among data. Therefore, a technology that binds data relationships is needed [6].

Unstructured data can also be managed by NoSQL DBMSs, which do not follow the relational storage strategy. NoSQL DBMSs have different approaches. Their common characteristic is that they are fundamentally scheme-free. The NoSQL DBMSs are classified into five categories: key-value stores, column-based stores, graph-based stores, object stores, and document-based stores [7]. Despite NoSQL DBMSs being excellent at dealing with many challenges associated with unstructured data. They also lack certain fundamental features where they provide scalability, but with increased data size, the scalability limits are slightly reduced. NoSQL DBMSs lack ACID properties because it is designed to run on a cluster of servers and personal computers. As a result, some organizations especially Google found that the use of NoSQL DBMS required their developers to spend a significant amount of time writing codes to process inconsistent data to increase productivity [8].

2.1.3 Semi-structured Data

Semi-structured data is characterized by a dynamic scheme. It is not constrained by a rigid structure. Many researchers proposed semi-structured data models that self-describe. They rely on data organization in labeled trees and query languages to access and update data. To express data exchange, these models propose various forms of data serialization. The semi-structured data models use a flexible structure to represent data. Some items may have missing attribute values, whereas others may possess additional attributes. An attribute may be complex or multivalued [9]. The most popular semi-structured data model is Extensible Markup Language (XML) [10,11]. Unfortunately, XML is not a convenient format for representing the data-oriented semi-structured data as it needs some rules to represent and process document-semi-structured data [12]. For oriented semi-structured data, JSON has become common for simple, integrated, and compared effective formatting treatment [13]. The evolution of JSON is approaching, along with NoSQL document databases such as MongoDB and Couch DB that provide original JSON stores [14,15]. Also, recent advances now in RDBMSs provide support for JSON data type [16].

2.2 Big Data Storage

2.2.1 Structured Data Storage

RDBMSs efficiently store structured data in a tabular format organized into tables consisting of columns and rows with relationships among the tables. They follow a predefined schema to define the type and structure of data and their relationships [17]. Some of the most common applications using relational databases with structured data are sales transactions, airline reservation systems, inventory control, and ATM activity.

2.2.2 Unstructured Data Storage

There is a dramatic increase in the use of videos and images in our daily life attract significant attention from researchers in various fields. Images and videos are unstructured data that need security and privacy such as medical images. Therefore, many researchers introduced a higher imperceptibility schema, better robustness against watermarking attacks and extraction of medical images in medicine applications [18,19].

This study focuses on videos storage as an example of unstructured data. The video storage strategy could be one of the following:

File system-based video storage, large object (LOB) database video data storage based on data type, or a hybrid solution known as “data links”.

The first video storage strategy involves storing videos on a file system. This strategy works very well with large video files [20]. However, there are still some drawbacks when supporting video storage using this strategy. It is difficult to format videos with their metadata inconsistency because the content files are located outside the database. Many video thumbnails and video content descriptions in the directory also result in an ineffective response to the data request. Thus, the implementation of the file system strategy remains a challenging task.

The second video storage strategy involves storing videos in databases rather than the file system. It stores video data in the binary large object (BLOB) field of the database. There are several features for storing videos in DBMSs, such as the following [21]:

- The data take advantage of DB security mechanisms and reduces maintenance costs.
- Data retrieval from the DB database is efficient.
- Management and access control are facilitated.
- A DBMS ensures data consistency, especially regarding original metadata preservation, often considered an issue in file systems.

On the other hand, using DBMS approaches for video storage and access is time-consuming and may increase DBMS load, especially with large videos. However, reaching a maximum storage capacity in DBMS does not necessitate using a more powerful system to handle more videos. A different approach to load balancing in the background is required to enable the DBMS to handle more data.

The third video storage strategy is called "data links". It stores videos in the file system and coordinates the video files and their metadata with consistency for transactions using the database [22].

2.2.3 Semi-Structured Data Storage

This study presents an approach to represent the multivalued attributes and M:N relationships that are semi-structured data. The approach uses the JSON data model to create a flexible schema and to store semi-structured in RDBMS. JSON is widely used to store and exchange semi-structured data [23]. It was originally defined by Douglas Crockford and was designed primarily for data serialization and transmission over a network. The JSON data model consists of key/value pairs where the key is a string, and the value can be of any JSON data type (object, array, string, number, Boolean, and null) [24]. For example:

```
{"Name": "smith",
"Age": 46,
"Indicted": True,
"Car": null,
"phones":[{"number":"110–122–132"}, {"number":"222–111–222"}],
"Job": {"Company": "advertisement", "address": "Stafford"}}
```

JSON objects can be nested inside other objects as in the previous example. Several attempts have been made to develop a suitable solution for creating a dynamic schema in DBMSs using JSON. The most popular relational databases adopted it as a native column type [25,26]. One of the first ideas to store and query JSON is Argo [27] where two approaches were proposed to store JSON objects. The first approach was employed to store JSON objects in one large table with OID, keys, and three columns for the data type: string, number, and Boolean. The second approach was used to store a JSON object in three tables, each with separate types. Argo introduced Argo/SQL, a query language for querying JSON objects. Paper [27] also introduced a NoBench benchmark for running a set of queries over JSON objects. Argo was implemented in two RDBMSs (MySQL and PostgreSQL) and compared with a NoSQL DBMS, MongoDB using the NoBench benchmark. The running of Argo on

RDBMSs makes its performance high enough to make it a very convincing substitute for MongoDB for small data. Liu et al. present supports JSON as a native type in the Oracle DBMSs and the results indicated that indexed JSON is faster than other solutions [28]. Liu et al. also proposed a new query JSON format namely OSON released in Oracle [29].

3 The Proposed Approaches

The research presents two optimization approaches to DBMS storage. The first proposes an efficient approach for video storage in RDBMSs. The second approach uses the JSON to represent multivalued attributes and many to many relationships in RDBMSs. The approaches were tested on a workstation with a CPU speed of 2.20 GHz (two processors), memory capacity of 32.0 GB, and Windows Server 2016 standard.

3.1 The First Approach

This approach is designed to store videos using three phases, as presented in Fig. 1. The top phase consists of the user-side interface. The proposed interface supports video upload in different formats, delete and retrieve video using SQL query sent to the server for execution. Once a video has been selected, the video is played using a media player. To deal with the various needs of video database users, the interface provides multiple access points to any specific piece of data is required.

The second phase is extracting the descriptive video information using a python program developed for the experiment to extract the video metadata. After the values of the fields in the metadata are obtained, they are stored in the databases structures to facilitate the experiments. The video was retrieved using information like title, properties (duration, type, size, etc.)

The last phase splits the video into chunks with different size sets. The chunks are stored in rows in the database. A B-tree index ensures quick searches for the correct chunk number when performing random-access reads and writes. This enables efficient storage and fast retrieval of video.

Under these circumstances, videos are stored in two tables; Video _Metadata and Video _Chunks as presented in Tabs. 1 and 2 respectively. The Video _Metadata table is used to store video attribute information. The Video _Metadata attributes include id, title, duration, size, type, and insertion moment. "id" stands for the unique video identifier, "title" stands for the name of the video, "duration" stands for the video duration in minutes, "size" stands for the video size in MB, "type" stands for the video format and "insertion moment" stands for the time when the video is uploaded to the database. The Video _Chunks table is used to store the chunks of videos. The Video_ Chunks table attributes include _id, video _id, n, and data. "_id" stands for the unique file chunk identifier, "video _id" is the same as "id" in the Video _Metadata table, "n" stands for the relative order in the video chunks, and "data" stands for the video chunk loaded into the database as BLOB data. Tab. 1 contains the video metadata with only one primary key, whereas Tab. 2 stores the BLOB accompanied with a foreign key to the Video _Metadata table. As a result, when the user needs to access only a specific set of video bytes, these chunks are only imported to memory. This is extremely helpful when dealing with large amounts of media content that must be selectively read or edited. Splitting the attributes into two tables is important as searching operations based on metadata can be efficiently performed without interacting with the other attributes that contain actual binary video data.

3.1.1 Case Study

The experiment was conducted on a dataset of different videos of various sizes and formats. In the first phase, the videos were prepared for uploading using the user interface. The videos were then

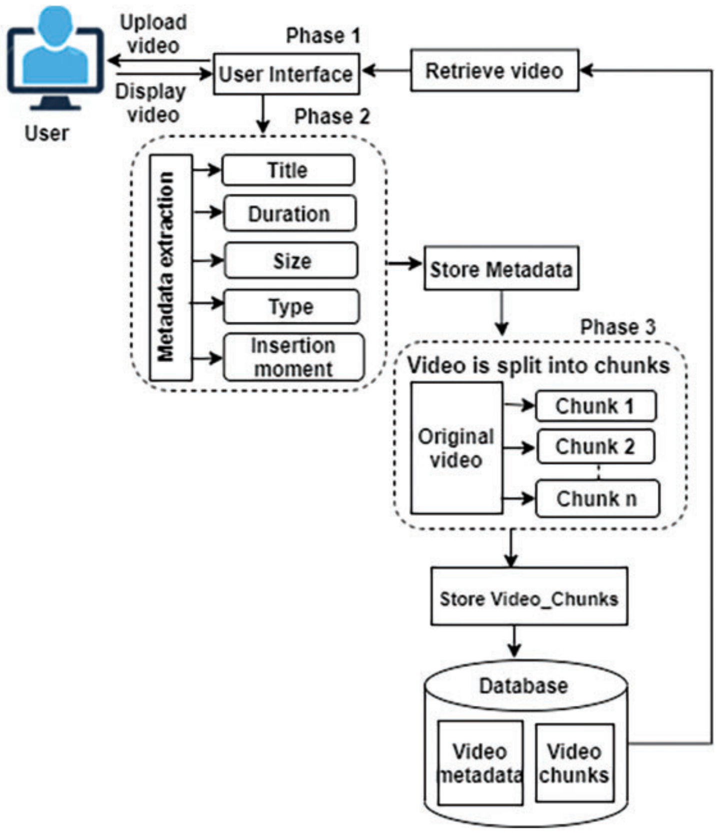

Figure 1: The proposed approach phases

Table 1: Video_metadata

id	title	duration	size	type	insertion moment

Table 2: Video_chunks

_id	video_id	n	data

segmented into a number of chunks with varying fragment size sets. The chunks were obtained and loaded into the database. The approach was employed with the open-source DBMS MySQL 8.0 as all the videos used the MySQL features to handle binary data. The approach was proposed to enhance the MySQL approach for storing videos. So, the proposed approach will be known as E-MySQL. To assess E-MySQL efficiency, it was compared with two open-source DBMSs: MySQL (an RDBMS), and MongoDB (a NoSQL DBMS). It should be mentioned that, in MySQL, the video is stored as a binary object using the BLOB data type, and its metadata is stored in the same table containing the video. While MongoDB stores binary data such as images, audios, and videos through the lightweight file storage specification called GridFS. GridFS is a type of file system that divides the large file into data chunks and each data chunk is stored in a separate document [30].

3.1.2 Results and Analysis

The practical performance aspect is storage and retrieval efficiency, how much data can be stored in memory and quickly retrieved. Therefore, a discussion on performance between databases and the comparability of performance must be taken. In this subsection, the performance of the proposed approach is evaluated.

a) Video Storage

The performance analysis put the main concentrate on the speed of inserting data. The tests make insert queries with different video sizes to compare the DBMSS run time. The time taken to insert and store the video data into databases is measured by seconds. To calculate the speed of the databases, it fetches the first time before executing the insert operation. When the insert operation ends, it fetches the second time. The second time minus the first time is the time needed representing the total time to execute the insert operation. A comparison among MySQL, MongoDB, and E-MySQL is presented in Tab. 3. According to the results, the time required to store video data in databases increases in conjunction with the size of the video. Furthermore, MongoDB and MySQL DBMSs consume more time than E-MySQL when storing the same video data, as illustrated in Fig. 2. It is also obvious that the use of MySQL to handle videos will consume much time, and the database will stop, especially with large videos. For example, MySQL stops working when inserting a video larger than 60 MB. To sum up, E-MySQL is becoming more efficient with the increase in the number and size of videos in RDBMSs.

Table 3: Video insertion time in databases

Size (MB)	MySQL	MongoDB	E-MySQL
0.531	0.5625	0.1643	0.0625
1.59	0.5782	0.1944	0.0881
2.39	0.6875	0.2304	0.1875
7.74	1.0157	0.7445	0.5872
21.17	2.4534	1.4466	0.9442
26.35	2.844	1.704	1.0325
35	3.7191	2.3906	1.2896
44	4.7349	2.9526	1.9813
60.7	7.5634	4.0276	2.0387
83	$***$	5.6957	2.2351
119	$***$	8.3708	2.7501
224	$***$	14.609	3.502

Note: $***$means stopping the MySQL for large sizes of videos.

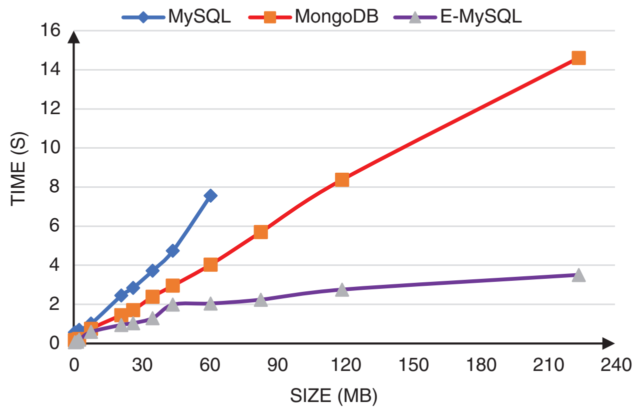

Figure 2: Video insertion time in databases

b) Video Retrieval

E-MySQL pays attention to the speed of reading and writing data from/into the database. The performance while performing the select operation is evaluated. There are two ways to retrieve video in E-MySQL:

- The user can request video via the database interface.
- The user can remotely access the video using the proposed interface, enabling remote invocations to read and write data into the database.

E-MySQL reassembles chunks as needed when queried for a video. It is even possible to access data from the metadata table bypassing video. To retrieve the video through the Video _Metadata table, the user specifies query terms and the "id" value. Because the relationship between "id" in Video _Metadata table and "video _id" in Video _Chunks table is equal. Increasing the retrieval speed is an important criterion for any video retrieval database system. Therefore, the efficiency of the proposed is evaluated by computing the retrieved time of the video. The access in experiments has been conducted by retrieving videos of different sizes. Query execution was used to evaluate the access performance of the videos and metadata. The results indicated that the time required to access the video data in E-MySQL increased in conjunction with the size of the video. This is in line with the other two DBMSs.

Hence, through this experiment, an effort is made to enhance the capability of MySQL to handle unstructured data such as videos. E-MySQL guarantees optimal performance in both storage and access. Also, it achieves scalability of data storage where the database supports a large amount of data to be stored. Because the size of the video impacts the performance, the process of splitting data across multiple records allows each record to host a subset of video data and curtail database load. The proposed storage makes the storage of videos with large sizes possible and effective. Furthermore, E-MySQL is useful for storing any other files where access to file parts is required without loading the entire file into memory.

3.2 The Second Approach

Modeling is considered one of the most important steps in developing database systems. The JSON data model is currently the most modern knowledge representation, and it is used in most NoSQL DBMSs. On the other hand, the Entity/Relationship Model (E/RM) is a conceptual model widely used in RDBMSs. This approach employs the JSON model to represent some data in E/RM in RDBMS. As illustrated in Fig. 3, the approach starts with designing the Entity/Relationship Diagram

(E/RD) that gives a good overview of the entities and the relationships. The E/RD is mapped to the relational model [31] or the proposed approach according to multivalued attributes and M:N relationships.

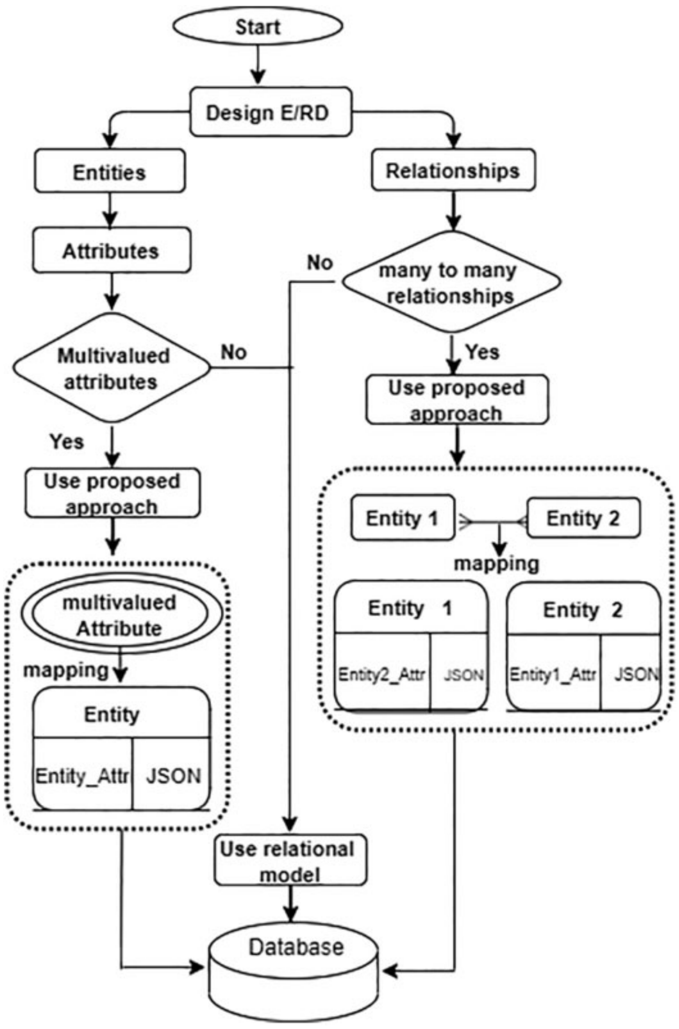

Figure 3: Flowchart of the proposed approach

The following types can represent the attributes in the relational model: simple attributes, composite attributes, stored attributes, derived attributes, single-valued attributes, and multivalued attributes. Except for the multivalued attribute, which is mapped to new relation, all of these types are mapped to a column in the relation. The relationships between entities in relational model mapping always work with two relations at a time. The first relation is called the primary or parent relation and the second is called the related or child relation. While the M:N relationships are mapped by using three relations. The third relation has the primary keys (eventually the relationship attributes, if any) of those original relations. The proposed approach is still relational while the JSON format maps multivalued attributes. In addition, to avoid creating new relation and attribute redundancy, the approach represents the M:N relationships by JSON format using the Two Way Embedding (TWE) method. It duplicates one side of the 'many' entities in the relationship [32]. A case study will be presented here to demonstrate this approach.

3.2.1 Case Study

The approach is applied to the company relational schema presented in [33] as a standard relational database schema. The experiments were executed by mapping multivalued attributes and M:N relationships. The storage difference and query performance between the relational model and this approach in the relational database are analyzed. The schema of the proposed approach contains many details related to the "company employee" as shown in Fig. 4. It consists of four relations: employee, project, department, and dependent. There is an M:N relationship between the employee and the project. It represents all of the employee's projects and the hours worked. This relationship is mapped using the JSON format by adding the "pro_works_on" attribute in the employee relation and the "emp_works_on" attribute in the project relation. The "pro_works_on" represents all employee project numbers "p_no" and hours. The JSON representation is ([{"p_no": value, "hours": value}, {"p_no": value, "hours": value} . . .]). The "emp_works_on" attribute represents all employee identifiers "ssn" and hours. The JSON representation is ('[{"ssn": value, "hours": value}, {"ssn": value, "hours": value} . . .]'). The department relation contains the "dlocation" attribute representing the departments' location. "dlocation" is a multivalued attribute; thus, it maps using the JSON format. The JSON representation for this attribute is '[value1, value2, value3]'.

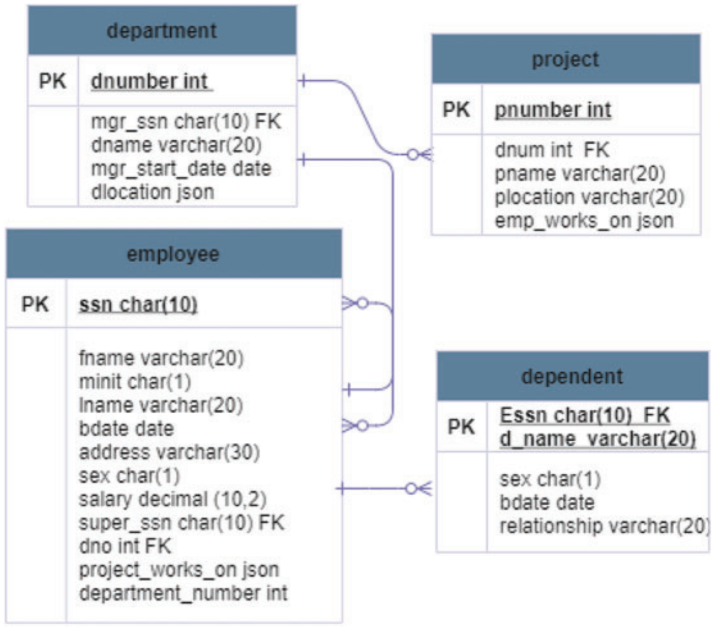

Figure 4: Schema of the proposed approach

3.2.2 Results and Analysis

It was found that MySQL is an appropriate RDBMS according to the DB Drive ranking [34]. Therefore, it was applied in this approach. MySQL has supported JSON as a data type since Version 5.7 [35–43]. It supports the manipulation of JSON objects with the operators (-> and ->>). The two operators can be used to extract data from JSON columns. MySQL also provides a path that begins with a dollar sign ($) to access the nested objects. This approach will be called A-MySQL in this study. The A-MySQL experimental results were obtained and compared with the traditional relational model in MySQL. To evaluate A-MySQL with JSON, a dataset has been created for different sizes of records (10 K, 100 K, 250 K, 500 K, 750 K, 1 M, 2 M, and 3 M) and two aspects are measured. First, the space required to insert the dataset of different sizes. Second, the time required to insert the dataset has been measured for the most representative sizes (10 K, 100 K, and 1 M). Furthermore, four queries with different complication levels were chosen to examine all of the schema properties thoroughly. Tab. 4 represents the consuming storage space in MySQL and A-MySQL. These results indicated that A-MySQL reduced the storage space required to store data, as illustrated in Fig. 5; therefore increasing the performance.

Table 4: Size of data on the disk

	MySQL	A-MySQL
10K records	5.2 MB	4.1 MB
100K records	30.2 MB	21.4 MB
250K records	88.3 MB	49.2 MB
500k records	152.3 MB	96.2 MB
750K records	234.6 MB	145.3 MB
1M records	328.3 MB	203.3 MB
2M records	412.3 MB	281.5 MB
3M records	498.8 MB	372.9 MB

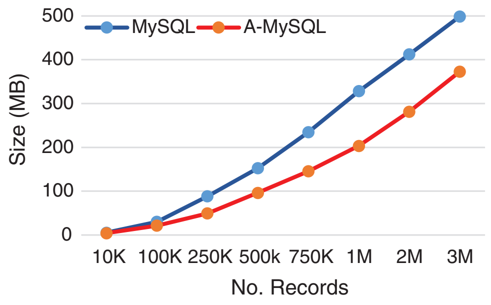

Figure 5: Storage space in MySQL and A-MySQL

Tab. 5 shows a change in the insertion speed significantly, where A-MySQL is faster than MySQL.

Table 5: The time needed to insert datasets into the database

Database	Number of records		
	10K records	100K records	1M records
MySQL	1.57812 s	15.46875 s	330.17187 s
A-MySQL	1.09375 s	10.671875 s	128.07812 s

The performance of the approach is evaluated to carry out an effective comparison between the approach and the relational model. Therefore, different analytical queries to precisely examine all of the A-MySQL characteristics are prepared. A set of performance properties have been tested inclusive of queries with attribute sets, queries with numeric aggregations, and queries with attribute join via entities. No indexes are created for the JSON data to ensure that any differences in the efficiency of the SQL/JSON query processing result from the different encoding formats. The SQL representation of the queries in MySQL and A-MySQL is presented in Tab. 6. The first query is to retrieve attributes based on the value of a specified attribute. The query's goal is to examine the access performance to the JSON attributes. The query selects all project numbers, hours worked by an employee, and his/her salary. In the second query, the department numbers are listed and the number of locations of each department is counted. Therefore, this query can examine the aggregation function performance for the objects within the A-MySQL. The third query can list the "Ssn" numbers of the employee working at a project named "Sirius". This is a more complex query to extract data from a JSON document partially. The fourth query uses the second and third queries altogether.

The four different queries were executed and registered the time for query retrieval at different times. The times required to execute these queries using the platform mentioned above are presented in Tab. 7. It is clear that A-MySQL's execution time is faster than MySQL and slightly increases with the data size as presented in Fig. 6. The objective of these queries was not to prove absolute performance numbers that can be obtained but to assess the proportional performance properties of each approach. Using the JSON format to present dynamic data, A-MySQL can execute a set of objectives. The use of A-MySQL can increase readability and simplify the creation of queries by reducing the number of needed joins.

Table 6: The SQL representation of the queries in MySQL and A-MySQL

Query	MySQL	A-MySQL
Q1	SELECT pnumber, hours, salary FROM works_on, employee where employee.ssn = works_on.ssn and ssn=111111191;	SELECT project_works_on, salary FROM employee where ssn=1111111191;
Q2	SELECT dnumber, count(d_location) FROM dept_loc group by(dnumber);	SELECT dnumber,JSON_length(d_location) FROM department
Q3	SELECT ssn FROM works_on, project where project.pnumber = works_on.pno and pname = 'Sirius';	SELECT emp_works_on->'$ [0].ssn',emp_works_on->'$ [1].ssn ' FROM project where pname = 'Sirius';
Q4	SELECT sum(hours) as total_hours FROM works_on group by (ssn);	SELECT (sum (project_works_on->'$ [0].hours ') + sum (project_works_on->'$ [1].hours ')) as total_hours FROM employee group by(ssn)

Table 7: The execution times of Tab. 6 queries

No. records	MySQL				A-MySQL			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
10 k	0.062	0.062	0.078	0.125	0.047	0.031	0.015	0.015
100K	0.468	0.218	0.109	0.188	0.218	0.047	0.046	0.047
250K	0.796	0.796	0.127	0.978	0.718	0.062	0.062	0.062
500k	0.859	0.859	0.344	1.317	0.797	0.11	0.074	0.078
750k	1.373	1.373	0.385	2.524	1.062	0.148	0.09	0.141
1M	2.282	2.282	0.591	3.091	1.547	0.187	0.125	0.172
2M	3.812	3.812	0.730	3.834	1.832	0.215	0.195	0.29
3M	3.843	3.843	1.041	4.01	2.021	0.271	0.214	0.473

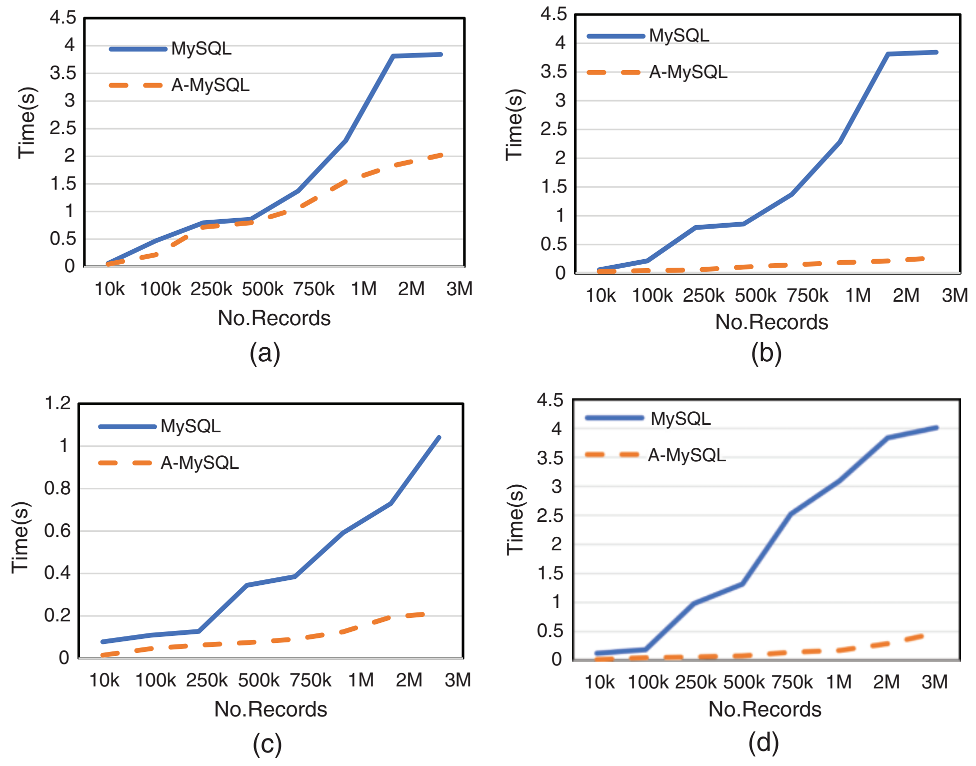

Figure 6: Execution time as a function of the number of records obtained for test queries. (a) Execution time of Q1, (b) Execution time of Q2, (c) Execution time of Q3, (d) Execution time of Q4

4 Conclusion

The relational model needs to be enhanced to handle a variety of data. This research focused on improving MySQL storage performance. The article introduced two approaches to handle unstructured and semi-structured data. The first approach improved storage and retrieval videos (unstructured data). The result of the performance comparison of this approach with those of MySQL RDBMS, and MongoDB NoSQL DBMS revealed that the proposed approach enabled RDBMSs to store video into the database more efficiently. Also, it speed up the in-database video access and improved database scalability so the performance was boosted. The second approach solved a dynamic schema store in the RDBMS. The approach used JSON format to represent multivalued attributes and M:N relationships. It has been compared to the relational model in MySQL. Results evaluation on a large dataset illustrated that using JSON format in a relational database is even more effective than the relational model in storing and retrieving semi-structured data. This approach helps software developers in choosing the appropriate data model when designing an application that uses semi-structured data. The two approaches add big data features to RDBMSs that allow users to manage structured, unstructured and semi-structured data in a single database engine.

Further Work: In the future, additional big data features may be added to RDBMS to incorporate more NoSQL DBMS capabilities. For instance, handling a huge load of data and supporting queries against non-relational data stored in RDBMSs therefore the ability to view and analyze data.

Funding Statement: This research was supported by a grant of the Korea Health Technology R&D Project through the Korea Health Industry Development Institute (KHIDI), funded by the Ministry of Health & Welfare, Republic of Korea (Grant Number: HI21C1831) and the Soonchunhyang University Research Fund.

Conflicts of Interest: The authors declare no conflict of interest regarding the publication of this paper.

References

[1] M. P. Bach, Ž. Krstić, S. Seljan and L. Turulja, "Text mining for big data analysis in financial sector: A literature review," *Sustainability*, vol. 11, no. 5, pp. 1277, 2019.

[2] S. Praveen, U. Chandra and A. A. Wani, "A literature review on evolving database," *International Journal of Computer Applications*, vol. 162, no. 9, pp. 35–41, 2017.

[3] D. D. de Macedo, A. Von Wangenheim and M. A. Dantas, "A data storage approach for large-scale distributed medical systems," in *2015 Ninth Int. Conf. on Complex, Intelligent, and Software Intensive Systems*, Santa Catarina, Brazil, pp. 486–490, 2015.

[4] T. Li, Y. Liu, Y. Tian, S. Shen and W. Mao, "A storage solution for massive IoT data based on NoSQL," in *2012 IEEE Int. Conf. on Green Computing and Communications*, Besancon, France, pp. 50–57, 2012.

[5] S.-Y. Choi and K. Chung, "Knowledge process of health big data using MapReduce-based associative mining," *Personal and Ubiquitous Computing*, vol. 24, no. 5, pp. 571–581, 2020.

[6] H. Jung and K. Chung, "Knowledge-based dietary nutrition recommendation for obese management," *Information Technology and Management*, vol. 17, no. 1, pp. 29–42, 2016.

[7] K. A. ElDahshan, A. A. AlHabshy and G. E. Abutaleb, "Data in the time of COVID-19: A general methodology to select and secure a NoSQL DBMS for medical data," *PeerJ Computer Science*, vol. 6, no. 3, pp. e297, 2020.

[8] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost *et al.,* "Spanner: Google's globally distributed database," *ACM Transactions on Computer Systems*, vol. 31, no. 3, pp. 1–22, 2013.

[9] L. Ramalho, "From ISIS to CouchDB: Databases and data models for bibliographic records," *Code4Lib Journal*, vol. 12, no. 13, pp. 1–15, 2011.

[10] R. Bača, M. Krátký, I. Holubová, M. Nečaský, T. Skopal *et al.,* "Structural XML query processing," *ACM Computing Surveys (CSUR)*, vol. 50, no. 5, pp. 1–41, 2017.

[11] Z. Brahmia, H. Hamrouni and R. Bouaziz, "XML data manipulation in conventional and temporal XML databases: A survey," *Computer Science Review*, vol. 36, no. 4, pp. 100231, 2020.

[12] C.-O. Truică, E.-S. Apostol, J. Darmont and T. B. Pedersen, "The forgotten document-oriented database management systems: An overview and benchmark of native XML DODBMSes in comparison with JSON DODBMSes," *Big Data Research*, vol. 25, no. 12, pp. 100205, 2021.

[13] D. Durner, V. Leis and T. Neumann, "JSON tiles: Fast analytics on semi-structured data," in *Int. Conf. on Management of Data (SIGMOD'21)*, New York, NY, USA, pp. 445–458, 2021.

[14] G. Harrison and M. Harrison, "MongoDB architecture and concepts," in *MongoDB Performance Tuning*. Berkeley, CA: Apress, pp. 13–32, 2021. https://doi.org/10.1007/978-1-4842-6879-7_2.

[15] S. A. Razoqi, "Data modeling and design implementation for couchDB database," *AL-Rafidain Journal of Computer Sciences and Mathematics*, vol. 15, no. 1, pp. 39–55, 2021.

[16] A. H. Alsup, "Examining the relationship between query performances when using different data models within relational database systems," Ph.D. dissertation, Colorado Technical University, 2021.

[17] R. Sint, S. Schaffert, S. Stroka and R. Ferstl, "Combining unstructured, fully structured and semi-structured information in semantic wikis," *CEUR Workshop Proceedings*, vol. 464, no. 12, pp. 73–87, 2009.

[18] X. R.Zhang, X. Sun, X. M.Sun, W. Sun and S. K. K.Jha, "Robust reversible audio watermarking scheme for telemedicine and privacy protection," *CMC-Computers Materials & Continua*, vol. 71, no. 2, pp. 3035–3050, 2022.

[19] X. R. Zhang, W. F. Zhang, W. Sun, X. M. Sun and S. K. Jha, "A robust 3-D medical watermarking based on wavelet transform for data protection," *Computer Systems Science & Engineering*, vol. 41, no. 3, pp. 1043–1056, 2022.

[20] R. Sears, C. Van Ingen and J. Gray, "To blob or not to blob: Large object storage in a database or a filesystem?," arXiv preprint cs/0701168, 2007. https://arxiv.org/ftp/cs/papers/0701/0701168.pdf.

[21] J. Azemović and D. Mušić, "Comparative analysis of efficient methods for storing unstructured data into database with accent on performance," in *2010 2nd Int. Conf. on Education Technology and Computer*, Shanghai, China, 1, pp. 403–407, 2010.

[22] S. Bhattacharya, C. Mohan, K. W. Brannon, I. Narang, H.-I. Hsiao *et al.,* "Coordinating backup/recovery and data consistency between database and file systems," in *Proc. of the 2002 ACM SIGMOD Int. Conf. on Management of data*, New York, NY, USA, pp. 500–511, 2002.

[23] L. Jiang and Z. Zhao, "JSONSki: Streaming semi-structured data with bit-parallel fast-forwarding," in *Proc. of the 27th ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA, 2022.

[24] P. Bourhis, J. L. Reutter and D. Vrgoč, "JSON: Data model and query languages," *Information Systems*, vol. 89, no. 8, pp. 101478, 2020.

[25] D. Petković, "JSON integration in relational database systems," *International Journal Computer Application*, vol. 168, no. 5, pp. 14–19, 2017.

[26] M. Piech and R. Marcjan, "A new approach to storing dynamic data in relational databases using JSON," *Computer Science*, vol. 19, no. 1, pp. 5–22, 2018.

[27] C. Chasseur, Y. Li and J. M. Patel, "Enabling JSON document stores in relational systems," *WebDB*, vol. 13, no. 4, pp. 14–25, 2013.

[28] Z. H. Liu, B. Hammerschmidt and D. McMahon, "JSON data management: Supporting schema-less development in RDBMS," in *Proc. of the 2014 ACM SIGMOD Int. Conf. on Management of data*, New York, NY, USA, pp. 1247–1258, 2014.

[29] Z. H. Liu, B. Hammerschmidt, D. McMahon, Y. Liu and H. J. Chang, "Closing the functional and performance gap between SQL and NoSQL," in *Proc. of the 2016 Int. Conf. on Management of Data*, New York, NY, USA, pp. 227–238, 2016.

[30] M. R. M. Chopade and N. S. Dhavase, "MongoDB, Couchbase: Performance comparison for image dataset," in *2017 2nd Int. Conf. for Convergence in Technology (I2CT)*, Mumbai, pp. 225–258, 2017.

[31] E. F. Codd, *The relational model for database management: Version 2*. Addison-Wesley Longman Publishing Co., Inc, Boston, MA, United States, 1990.

[32] L. Stanescu, M. Brezovan and D. D. Burdescu, "Automatic mapping of MySQL databases to NoSQL MongoDB," in *2016 Federated Conf. on Computer Science and Information Systems (FedCSIS)*, Gdansk, Poland, pp. 837–840, 2016.

[33] R. Elmasri and S. Navathe, *Fundamentals of database systems*. vol. 7. Pearson, University of Texas at Arlington, Texas, USA, pp. 115–193, 2017.

[34] DB-Engines Rankin, "Accessed Sept. 28, 2021," 2021. [Online]. Available: https://db-engines.com/en/ranking.

[35] M. Abdel Basset, D. El-Shahat, K. Deb and M. Abouhawwash, "Energy-aware whale optimization algorithm for real-time task scheduling in multiprocessor systems," *Applied Soft Computing*, vol. 93, no. 2, pp. 106349, 2020.

[36] M. Abdel-Basset, R. Mohamed, M. Abouhawwash, K. R. Chakrabortty and J. Michael, "EA-MSCA: An effective energy-aware multi-objective modified sine-cosine algorithm for real-time task scheduling in multiprocessor systems: Methods and analysis," *Expert Systems with Applications*, vol. 173, no. 1, pp. 114699, 2021.

[37] M. Abdel-Basset, R. Mohamed and M. Abouhawwash, "Balanced multi-objective optimization algorithm using improvement based reference points approach," *Swarm and Evolutionary Computation*, vol. 60, no. 3, pp. 100791, 2021.

[38] H. Seada, M. Abouhawwash and K. Deb, "Multiphase balance of diversity and convergence in multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 3, pp. 503–513, 2019.

[39] M. Abouhawwash and A. M. Alessio, "Multi objective evolutionary algorithm for PET image reconstruction: Concept," *IEEE Transactions on Medical Imaging*, vol. 40, no. 8, pp. 2142–2151, 2021.

[40] M. Abdel-Basset, N. Moustafa, R. Mohamed, O. Elkomy and M. Abouhawwash, "Multi-objective task scheduling approach for fog computing," *IEEE Access*, vol. 9, no. 3, pp. 126988–127009, 2021.

[41] M. Abouhawwash, "Hybrid evolutionary multi-objective optimization algorithm for helping multi-criterion decision makers," *International Journal of Management Science and Engineering Management*, Taylor & Francis, vol. 16, no. 2, pp. 94–106, 2021.

[42] S. T. Suganthi, A. Vinayagam, V. Veerasamy, A. Deepa, M. Abouhawwash *et al.,* "Detection and classification of multiple power quality disturbances in microgrid network using probabilistic based intelligent classifier," *Sustainable Energy Technologies and Assessments*, vol. 47, no. 4, pp. 101470, 2021.

[43] N. Mittal, H. Singh, V. Mittal, S. Mahajan, A. K. Pandit *et al.,* "Optimization of cognitive radio system using self-learning salp swarm algorithm," *Computers, Materials & Continua*, vol. 70, no. 2, pp. 3821–3835, 2022.