Auto JSON: An Automatic Transformation Model for Converting Relational Database to Non-relational Documents

K. Revathi[1], T. Tamilselvi[2], Batini Dhanwanth[3], M. Dhivya[4]
Department of Computer Science and Engineering, Panimalar Engineering College, Chennai, India[1, 2, 4]
Department of Computer Science and Engineering, Panimalar Institute of Technology, Chennai, India[3]

***Abstract*—In recent days, the demand for dealing large set of distributed data obsoletes the relational database and its structured query language (SQL) solutions in practice and paves the way for novel solutions in the name of non-relational database as not-only SQL (NoSQL). The NoSQL offers dynamic, flexible, scalable, highly available, greater performance and near real-time access to the distributed nature of voluminous data used for current industrial applications. Apart from these giant features of NoSQL, the SQL is still found to be in operation because of its popularity and standard. This paper projected an algorithm to convert the relational documents of MySQL into any document oriented NoSQL databases automatically without destructing the existing relational database setup and installing the NoSQL from scratch in the core machines. Java Script Object Notation (JSON) is a human readable data interchange format, being used in web development. The characteristics of JSON widened its use cases from web development to database storage. The Mongo DB, one of the most popular document oriented NoSQL adapts JSON format for its storage. The proposed algorithm is built based on its schema definition and the performance is captured through evaluating it against a sample database from hospital management system. The findings are discussed with great interest of addressing the challenges and revealing the scope for improvement.**

Keywords—Distributed data; document oriented NOSQL; hospital management system; Mongo DB; my SQL

I. Introduction

Databases Management System was created as a result of an increase in demand from industries for preserving their customer, stock, and account-related data and the process of obtaining relevant information from the data (DBMS). Systematic maintenance, as well as the effective storage and retrieval of data, are made possible by DBMS. Data are initially maintained using file systems, which store information directly in files without any connection to one another. Several models were presented as a result of the constraints in data access patterns that were found. International Business Machines (IBM) created a hierarchical model in 1960 that enables data organization in a tree structure by altering the parent-child relationship.

The network model, put forth by Charles Bachman in 1969, allows for the arrangement of data in a graph-like structure, with nodes serving as records and arcs as the connections between them. E. F. Codd created the relational model in 1970, which organizes data as tables and is currently regarded as a special model employed in important industries due of its features [1].

When dealing with situations in the real world, the data is contained in a single container referred to as an object. This begins the object-oriented database that has been in use since 1985 [2]. In order to create a new database known as an object relational database, the capabilities of relational databases and object-oriented databases were combined. By adding more dimensions to the data by displaying it as a cube, online analytical processing capability (OLAP) is offered in place of transaction processing.

Big data has replaced the traditional data that used to occur over a longer period of time in sectors that deal with data created every day. Even today's data volume is measured in petabytes or zettabytes, and 50% of the data are unstructured. The performance of the relational database, which exclusively processes structured data, is good for reasonable workloads but degrades as it is scaled up. Big data and analytical processing have made Relational Database Management Systems (RDBMS), which use Structured Query Language (SQL) to operate, ineffective [3]. Carlo Strozzi was the first to suggest Not Just SQL (NoSQL). This RDBMS was file-based and lacked a SQL interface. NoSQL, often known as non-relational databases, was first introduced in 2009 by Eric Evans [4, 5]. It is recognized as a promising database to handle massive data.

NoSQL is an alternative to RDBMS, facilitate mechanisms to store and retrieve enormous data in a distributed platform. The features on NoSQL over RDBMS are listed below.

- Horizontal Scalability – new nodes can be added to dynamic accommodate the storage requirements / requests
- Sharding – balances the workload distribution over the clusters in the distributed environment
- High Availability – due to its distributed nature, there is no single point of failure. Replication promises the high availability
- Better Throughput – offer better throughput to even high volume data than RDBMS
- Faster Performance – facilitate faster performance for big data than RDBMS

NoSQL is a popular language due to its enormous storage capacity as well as the following characteristics that set it apart from SQL.

- ACID Free: The acronym ACID stands for atomicity, consistency, isolation, and durability and supports the SQL transaction notion [6, 7]. NoSQL, a distributed database, provides improved data storage by relying on consistency but does not guarantee ACID properties.
- BASE: BASE stands for fundamentally available, soft state (data may vary over time), and finally consistency. It assures a high degree of availability through replication (no requirement to have identical copies in all nodes for all the time). In order to prioritize availability above consistency, eBay proposed this database design behavior, and NoSQL adopts it [6-9].
- CAP: Eric Brewer proposed the CAP theorem at a symposium on the fundamentals of distributed computing in 2000. It states that in network shared data systems, there is a trade-off between consistency, availability, and partition tolerance. BASE, which is a reverse notion of ACID and it is derived from the CAP theorem, is not feasible to hold up in distributed database, any two only achievable at time [6, 8] and depicted in Fig. 1.

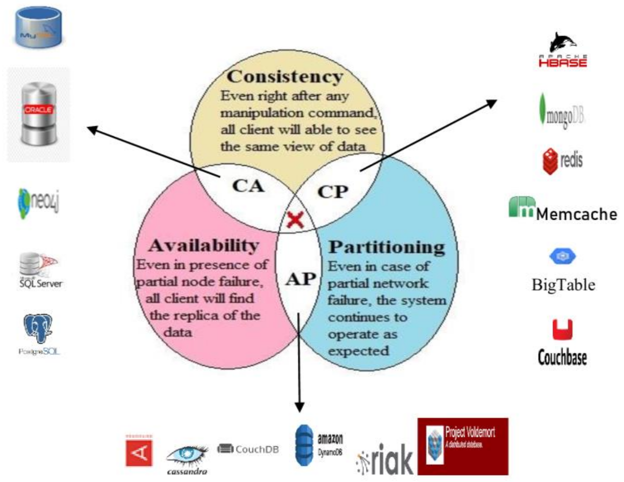

Consistency
Even right after any manipulation command, all client will able to see the same view of data
CA
CP
AP
Availability
Even in presence of partial node failure, all client will find the replica of the data
Partitioning
Even in case of partial network failure, the system continues to operate as expected
Memcache
BigTable
Couchbase
redis
mongoDB
HBASE
neo4j
SQL Server
PostgreSQL
CouchDB
cassandra
amazon DynamoDB
riak
Project Voldemort

Fig. 1. Visual representation of CAP theorem.

The rest of the paper is organized as follows: In Section II the significant past efforts made are analyzed and the basic detail about migration is discussed. The Section III discusses the migration algorithm in high interest. Section IV discloses the results obtained out of the device and Section V is intended to conclude the work.

II. Literature Review

A. Motivations for Migration Model

The latest and well-known initiatives to convert relational databases to NoSQL databases in order to address the emerging demand of modern applications have been thoroughly examined and are given in this section.

For the effective conversion of relational data into non-relational ones, the researchers offered a variety of methodologies, including data model-based, cloud-based, layer-based, web-based, and cross query engine based ones [10]. According to a research article by Liana Stanescu et al. [11], a base algorithmic view, the necessity of converting MySQL, a relational database, to a NoSQL database, nurturing the features of Mongo DB through addressing the fundamental principles to be followed in the transformation process, was elaborately captured.

Also, a framework was created using the NET platform, and an algorithm was created. The effectiveness of the algorithm was assessed based on the execution time of Create, Read, Update, and Delete (CRUD) operations over the databases provided with different workloads [12, 13]. The tuples are projected as documents, the columns are shown as fields in Mongo DB, and each MySQL database is represented as a collection. Using either embedding or referencing techniques, the key relationships in MySQL should be translated into Mongo DB.

Mahamood [14] described a method and created an interface using VB. Net that automatically converts Microsoft SQL Server tables to Mongo DB collections, in a manner comparable to the work mentioned with Liana Stanescu et al. [13].

Researchers Gyorodi, Kumar, Krishnan and Nair also conducted performance evaluations of MySQL and Mongo DB [15-18]. The results of the experiment demonstrated Mongo DB's superior efficiency. Additionally, the effectiveness is confirmed using the Yahoo Cloud Server Benchmarking (YCSB) tool by Kumar and Chakraborttii [19-20]. Saber et al. [21] discussed the efficiency of Mongo DB for handling Internet of Things (IoT) data in comparison to relational databases.

Singh [22] created a data conversion method that converts relational databases into Mongo DB collections and was discovered to be a successful pattern for cloud storage. In addition to the data transformation paradigm, Bajwa et al. [23] suggested data cleaning methods. A query-based transformation module to move from relational to non-relational data was developed by Al-Mahruqi et al. [24] and is designed to be used with apps.

B. Elemental Facts on Migration Model

The tables in MySQL databases are transformed into a collection of documents like in Mongo DB throughout the migration process, which is illustrated in Fig. 2 as a general process flow.

The mentioned transformation by Liana Stanestcu et al. [11-13] uses the metadata information of relational data bases and influences the Entity Relationship (ER) model, maps the key relations (1:1, 1: N, and M: N) found in RDBMS that are framed using primary and foreign keys against the relationship models in Mongo DB as embedding and referenced or linking.

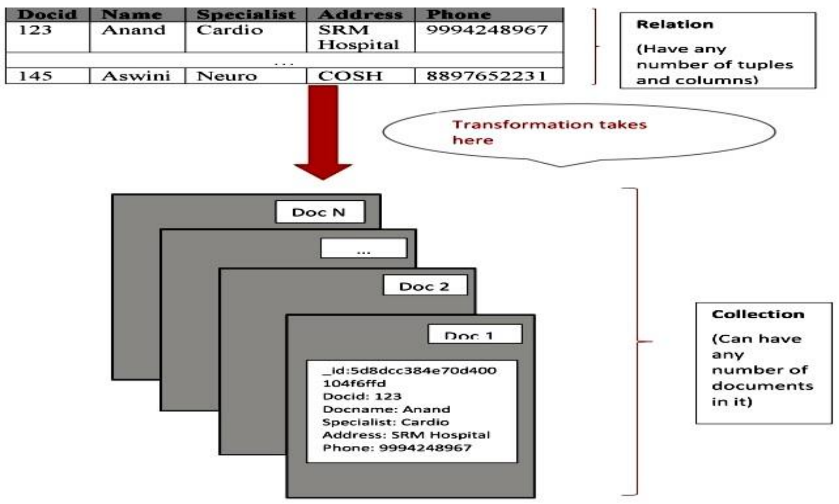
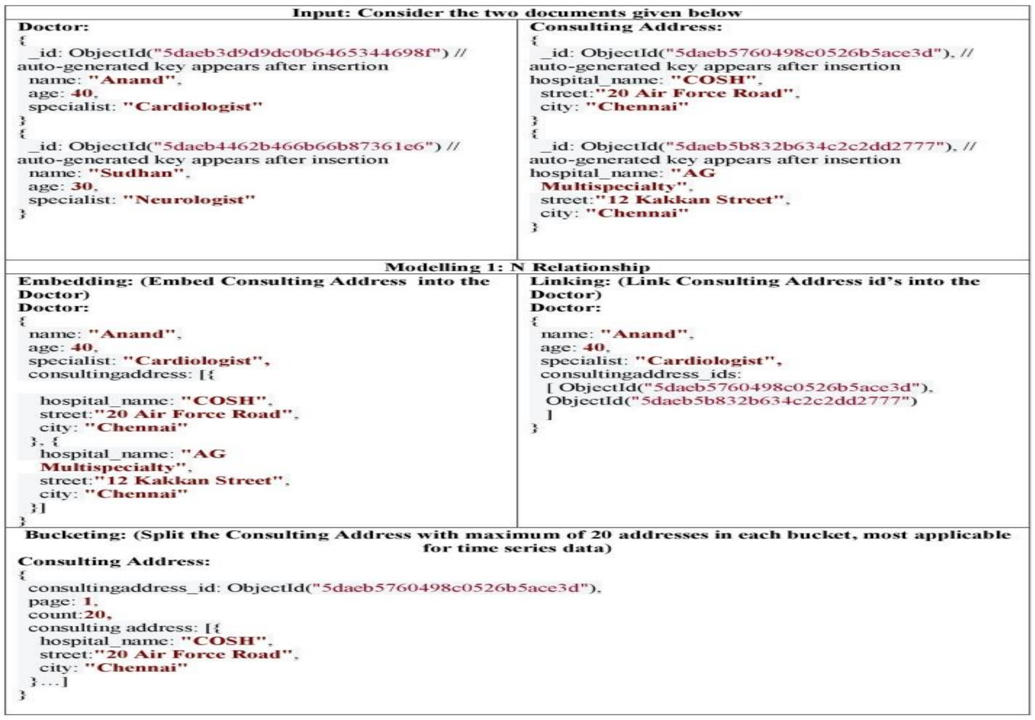


Fig. 2. The generic flow of migration process.

The following discusses in depth the various Mongo DB schema designs that might be used to reproduce the established RDBMS relationships and their recommended method of implementation [25, 26] by Gopinath et al. 2017, Yassine & Awad 2018.

In Mongo DB, there are only two modeling options for one-to-one (1:1) relationships: embedding or linking. When a document is embedded using arrays inside another document, all associated documents are shown as a single document. The embedding strategy increases data size, affects write performance, but makes retrieval simpler because it is combined with a single read. Although though documents are maintained separately, linking involves referencing one document's id through a field in another document using an automatically created key since relational databases store the data using foreign keys. In contrast to embedding, it shrinks the data yet affects read performance. Embedding is the best option because it offers effective retrieval.

There are three different techniques to model the one-to-many (1: N) relationship: embedding, linking, and bucketing. The first two still function. While bucketing is a third strategy that conforms to efficient retrieval while combining the advantages of embedding and connecting through slicing data into buckets with set data limits. Time series data applications benefit from bucketing. The methods for modeling the 1: N connection that was covered is depicted as a code snippet in Fig. 3.



Input: Consider the two documents given below

Doctor:
```
{
 _id: ObjectId("5daeb3d9d9dc0b6465344698f") //
auto-generated key appears after insertion
 name: "Anand",
 age: 40,
 specialist: "Cardiologist"
}
{
 _id: ObjectId("5daeb4462b466b66b87361e6") //
auto-generated key appears after insertion
 name: "Sudhan",
 age: 30,
 specialist: "Neurologist"
}
```

Consulting Address:
```
{
 _id: ObjectId("5daeb5760498c0526b5ace3d"), //
auto-generated key appears after insertion
hospital_name: "COSH",
 street:"20 Air Force Road",
 city: "Chennai"
}
{
 _id: ObjectId("5daeb5b832b634c2c2dd2777"), //
auto-generated key appears after insertion
hospital_name: "AG
 Multispecialty",
 street:"12 Kakkan Street",
 city: "Chennai"
}
```

Modelling 1: N Relationship

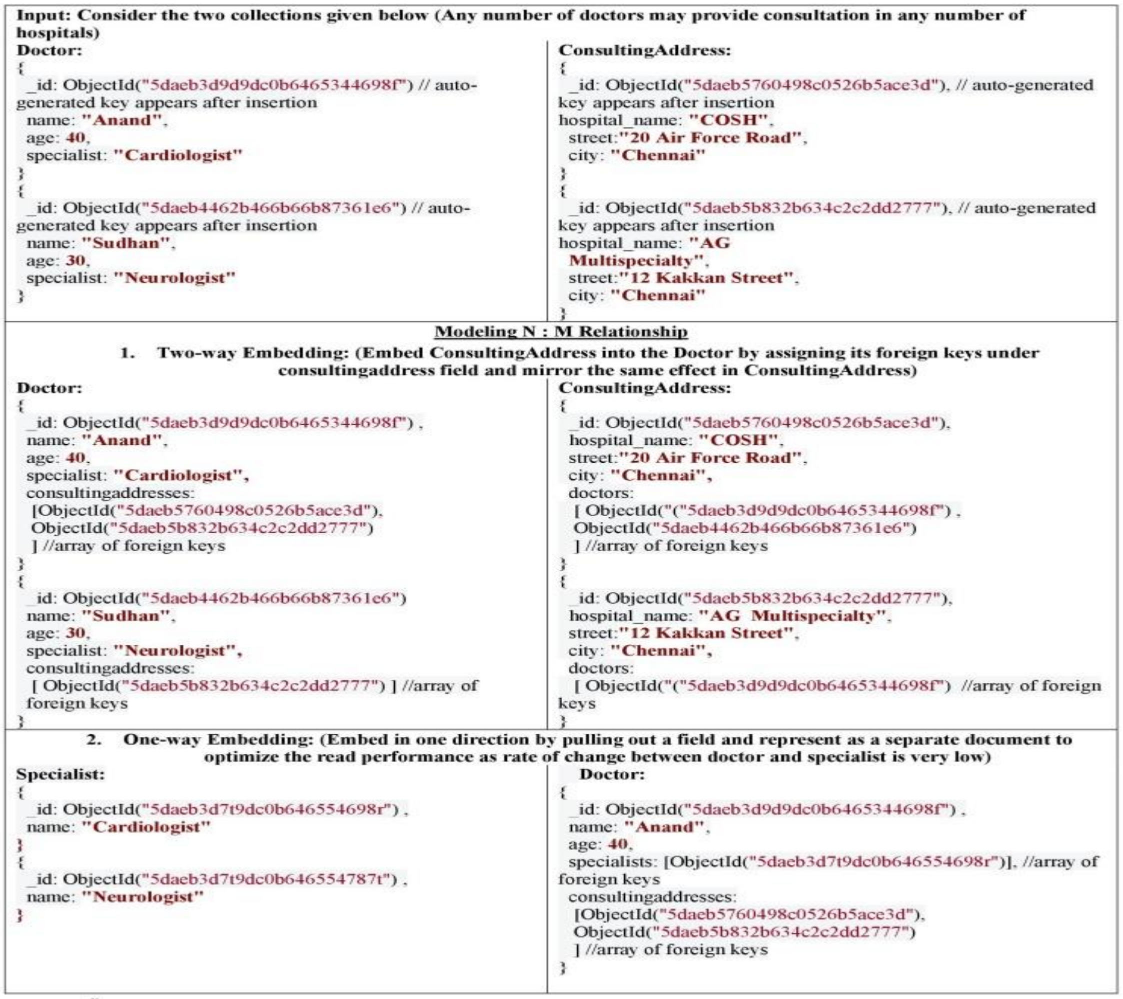
Embedding: (Embed Consulting Address into the Doctor)
Doctor:
```
{
 name: "Anand",
 age: 40,
 specialist: "Cardiologist",
 consultingaddress: [{

  hospital_name: "COSH",
  street:"20 Air Force Road",
  city: "Chennai"
 }, {
  hospital_name: "AG
  Multispecialty",
  street:"12 Kakkan Street",
  city: "Chennai"
 }]
}
```

Linking: (Link Consulting Address id's into the Doctor)
Doctor:
```
{
 name: "Anand",
 age: 40,
 specialist: "Cardiologist",
 consultingaddress_ids:
  [ ObjectId("5daeb5760498c0526b5ace3d"),
  ObjectId("5daeb5b832b634c2c2dd2777")
  ]
}
```

Bucketing: (Split the Consulting Address with maximum of 20 addresses in each bucket, most applicable for time series data)
Consulting Address:
```
{
 consultingaddress_id: ObjectId("5daeb5760498c0526b5ace3d"),
 page: 1,
 count:20,
 consulting address: [{
  hospital_name: "COSH",
  street:"20 Air Force Road",
  city: "Chennai"
 }...]
}
```

Fig. 3. Strategies to map 1: N relationship in Mongo DB.

The Fig. 4 illustrates two modeling approaches that are commonly used to represent many-to-many (M: N) relationships: two-way embedding and one-way embedding. In two-way embedding, one can insert one document into another by mirroring the foreign keys of the two documents in each field. When the size of the one-way embedding is extremely imbalanced, it is found to be useful as an optimal solution because it only enables embedding in one direction.



Input: Consider the two collections given below (Any number of doctors may provide consultation in any number of hospitals)

Doctor:

```
{
 _id: ObjectId("5daeb3d9d9dc0b6465344698f") // auto-
generated key appears after insertion
 name: "Anand",
 age: 40,
 specialist: "Cardiologist"
}
{
 _id: ObjectId("5daeb4462b466b66b87361e6") // auto-
generated key appears after insertion
 name: "Sudhan",
 age: 30,
 specialist: "Neurologist"
}
```

ConsultingAddress:

```
{
 _id: ObjectId("5daeb5760498c0526b5ace3d"), // auto-generated
key appears after insertion
hospital_name: "COSH",
 street:"20 Air Force Road",
 city: "Chennai"
}
{
 _id: ObjectId("5daeb5b832b634c2c2dd2777"), // auto-generated
key appears after insertion
hospital_name: "AG
 Multispecialty",
 street:"12 Kakkan Street",
 city: "Chennai"
}
```

Modeling N : M Relationship

1. Two-way Embedding: (Embed ConsultingAddress into the Doctor by assigning its foreign keys under consultingaddress field and mirror the same effect in ConsultingAddress)

Doctor:

```
{
 _id: ObjectId("5daeb3d9d9dc0b6465344698f") ,
 name: "Anand",
 age: 40,
 specialist: "Cardiologist",
 consultingaddresses:
  [ObjectId("5daeb5760498c0526b5ace3d"),
  ObjectId("5daeb5b832b634c2c2dd2777")
  ] //array of foreign keys
}
{
 _id: ObjectId("5daeb4462b466b66b87361e6")
 name: "Sudhan",
 age: 30,
 specialist: "Neurologist",
 consultingaddresses:
  [ ObjectId("5daeb5b832b634c2c2dd2777") ] //array of
 foreign keys
}
```

ConsultingAddress:

```
{
 _id: ObjectId("5daeb5760498c0526b5ace3d"),
 hospital_name: "COSH",
 street:"20 Air Force Road",
 city: "Chennai",
 doctors:
  [ ObjectId("("5daeb3d9d9dc0b6465344698f") ,
  ObjectId("5daeb4462b466b66b87361e6")
  ] //array of foreign keys
}
{
 _id: ObjectId("5daeb5b832b634c2c2dd2777"),
 hospital_name: "AG  Multispecialty",
 street:"12 Kakkan Street",
 city: "Chennai",
 doctors:
  [ ObjectId("("5daeb3d9d9dc0b6465344698f")  //array of foreign
keys
}
```

2. One-way Embedding: (Embed in one direction by pulling out a field and represent as a separate document to optimize the read performance as rate of change between doctor and specialist is very low)

Specialist:

```
{
 _id: ObjectId("5daeb3d7t9dc0b646554698r") ,
 name: "Cardiologist"
}
{
 _id: ObjectId("5daeb3d7t9dc0b646554787t") ,
 name: "Neurologist"
}
```

Doctor:

```
{
 _id: ObjectId("5daeb3d9d9dc0b6465344698f") ,
 name: "Anand",
 age: 40,
 specialists: [ObjectId("5daeb3d7t9dc0b646554698r")], //array of
foreign keys
 consultingaddresses:
  [ObjectId("5daeb5760498c0526b5ace3d"),
  ObjectId("5daeb5b832b634c2c2dd2777")
  ] //array of foreign keys
}
```

Fig. 4. Methodologies to map M: N relationship in Mongo DB.

C. Identified Research Gaps

The major efforts are for converting MySQL to Mongo DB was realized in [10-14, 22-26]. In most of the attempts, transformation mechanism was rooted from its schema design. The data type and key relationship exists between the tables plays an significant role in deciding the transformation strategy. Moreover the conversion is automated through an interface developed. In addition to key-relationship, table volume can be utilized in order to take precise decision about opting for embedding or linking. No existing work focused on time series data. By accounting current data type utilized in modern applications, the proposed work provided an automated database conversion solution for time series data.

III. AUTO JSON: A MIGRATION MODEL

The proposed work employs MySQL as a source database and selects Mongo DB as the destination NOSQL database based on inspiration drawn from related contributions listed in the preceding section. The proposed algorithm that automates the transformation of tables in source to collections in desired database is discussed in this section elaborating the data with the execution environment used for it.

A. Hospital Management System: A Source Database

A software program known as a hospital management system controls medical setup operations without the need of paper. The HMS incorporates all pertinent data, including those on doctors, patients, and related services. A hypothetical hospital administration scenario is used as the input database, and Fig. 5 depicts the appropriate relationships between the tables in the database.

Five tables make up the database, and connectors are used to create the relationship between the tables. Each class's primary key attribute is represented by the first entry in that class. It is beyond of scope to go into depth about class functions.

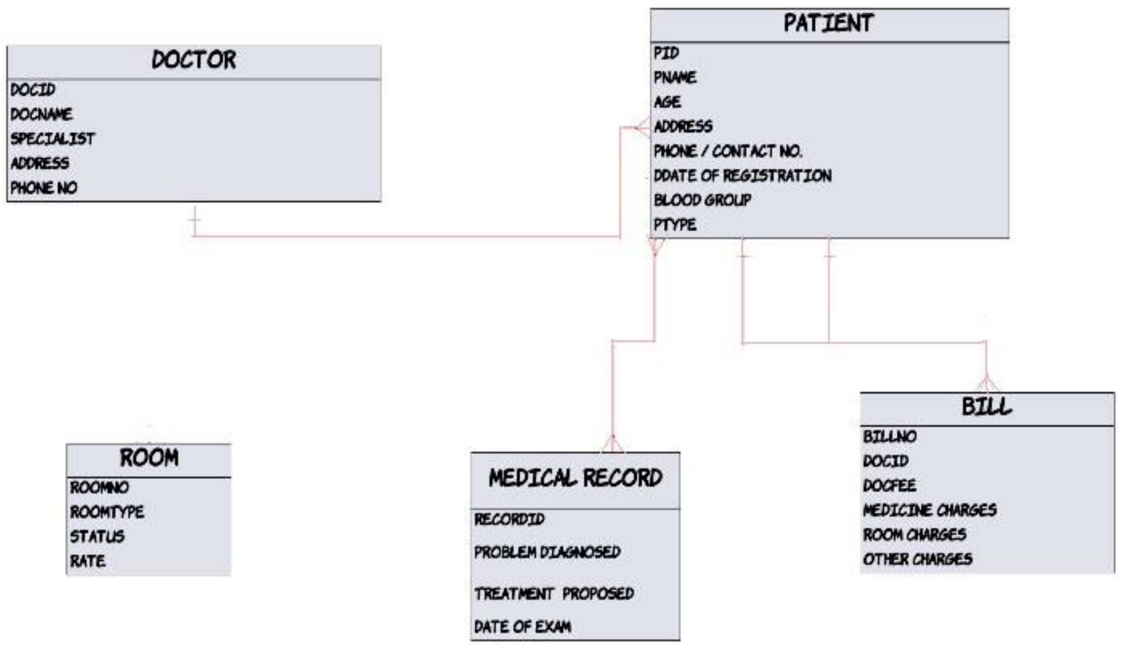


Fig. 5. A class diagram of typical HMS.

B. *Migration Algorithm*

Tables can be converted into collections using the suggested method's options for bucketing, embedding, linking, one-way embedding, and two-way embedding.

TABLE I. ALGORITHMIC STEPS FOR AUTO MIGRATION ALGORITHM

```
Algorithm 1: Auto JSON (SQL DB, Relationship _Mode,
Table_Volume, Data _Type)
Input: Relational Table (MySQL)
Output: JSON Documents (Mongo DB)
Extract meta data of  SQL DB
Observe the data pattern and relationship exists among the tables
 | if(Relationship_Mode == 1:1 && Table_Volume==High) then
 |        Perform Linking
 | else if (Relationship_Mode == 1: N && Data_type==
 |             Time_series) then
 |        Perform Bucketing
 | else if (Relationship_Mode == 1: N && Data_type!=
 |             Time_series) then
 |        Perform Linking
 | else if (Relationship_Mode == M: N  &&
 |             Table_Volume==High) then
 |        Perform Two Way Embedding
 | else if (Relationship_Mode == M: N  &&
 |             Table_Volume==Low) then
 |        Perform One Way Embedding
 | else
 |        Perform Embedding
Return the Collections in Mongo DB
```

The decision is significantly impacted by a number of factors, including table size, data type, and relationship method. These parameters are referenced about using meta-data that is given in the source database's information schema. Previous attempts focused on the sort of relation that already existed with keys between tables and left out any consideration for the data's nature or table volume. The proposed research optimizes the selection of transformation by taking these aspects into account, as shown in Table I.

The Extract, Transform, and Load (ETL) principle underlies the migration algorithm's operation. In order to convert tables into JSON collections, the meta-data of the table schema must first be extracted. Careful consideration of the schema information, such as field type, table volume, and relationship type, enables selection of options like linking, embedding, bucketing, and one-way or two-way embedding. Lastly, the collections are uploaded to a cloud storage system for further data analytics. Without installing the underlying database packages, the migration method automates the conversion of relational model tables to collections of non-relational model tables.

IV. RESULTS AND DISCUSSIONS

This section provides the details on the experimentation environment, evaluation methodologies and metrics to demonstrate the performance of the proposed migration strategy.

A. *Experimetation Setup*

The Auto JSON algorithm is coded as python program and executed on the Mongo DB ATLAS platform. Mongo DB ATLAS is a cloud database as a service (DBaaS) contributed by the Mongo DB. It has various provisions listed as follows.

- Provides all the features of Mongo DB
- Simplifies the automation process without overlooking the infrastructure, configuration of database, backups and so on.

- Ensures security and privacy.
- Facilitates choice of deploying the generated database in one of the platforms like Amazon Web Services (AWS), Google Cloud Platform (GCP) or Microsoft Azure.

B. Dataset and Queries for Evaluation

Evaluation is done using data that is in line with the hospital management system described in the preceding section. Compared to creation and read/retrieve, the scope for updating and deleting a record is minimal. The common users are restricted with the read-only and other security privileges. Hence, the following fundamental actions were picked to demonstrate the efficacy of the migration model that converts SQLDB to Mongo DB:

- Create with Insert
- Select

Among various DB functions, create function only allocates the space for data whereas the insertion command only populates the data into the space preserved by create. The delete command removes the entry from the space. Likewise the update command attempts to accommodate few changes on the data. The transformation task has its major focus on creating and presenting the same as JSON collections of Mongo DB. Here create function means a allocating a space provided with the data by means of insertion. Thus the comparative analysis of migration algorithm from MySQL and Mongo DB is captured by means of projecting the efficiency with respect to the basic operations named Create and Select.

The steps taken to record the performance of the suggested migration model is listed as follows:

- Register for a Mongo DB ATLAS user account.
- Put in the required libraries to assist migration.
  - pip install jsonmerge
  - pip install pymongo
  - pip install sqlalchemy
- Use Python to create the SQLDB code .
- Transform to Mongo DB Collection by running the migration algorithm in python as illustrated in Fig. 6.

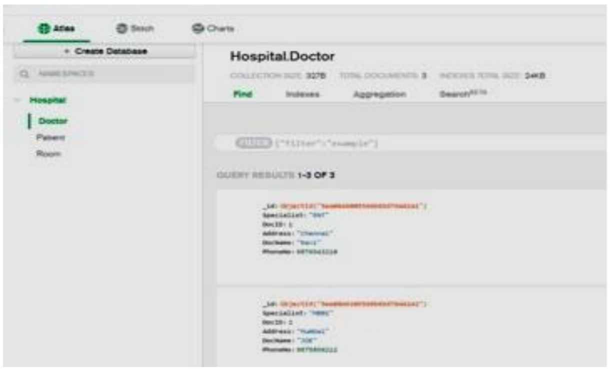


Fig. 6. Screenshot of completion of migration.

C. Performance Evaluation

The primary objective of the proposed work is to improve the efficiency in terms of execution time and memory utilization. In general execution or response time with respect to the database operations is calculated by accounting the time of observing the result set from the query initiation time. As aligned with this, the formula to evaluate response time is given in Eq. (1).

$$R_T = Q_C - Q_I \qquad (1)$$

Where R_T is the response time, Q_C is for query completion time and Q_I is known as query initiation time. In this case, the query is simply initiated by selecting a transformation operation, and it is completed by collecting the appropriate result set as collections. Since the source database contains non-time series data, bucketing is no longer within the purview of this project. The Fig. 7 displays the execution time plot of transformation operations.

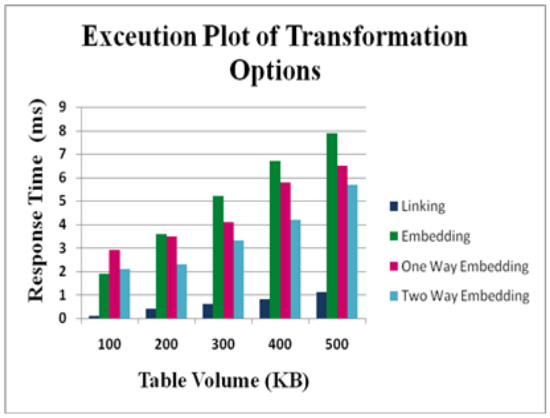


Fig. 7. Execution plot of transformation operations.

The next important metrics is demonstrating the effective memory utilization of the proposed algorithm. This is derived by invoking a comparison over consumption of memory against source and destination database. The effective memory utilization of proposed algorithm is formulated in Eq. (2) and pictorial representation is illustrated in Fig. 8.

$$MEU = (MC / MR) * 100 \qquad (2)$$

Where M_{EU} stands for effective memory utilization, M_C denotes memory consumptions observed in Mongo DB collections and M_R gives memory consumptions observed in MySQL relations.

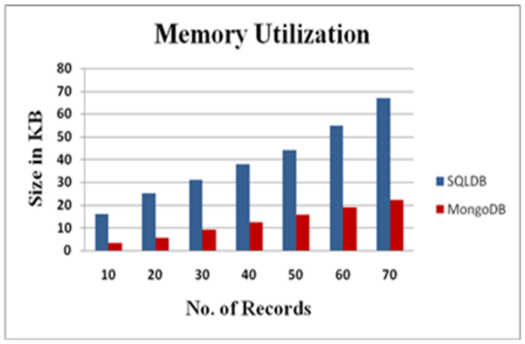


Fig. 8. Comparative analysis of memory usage.

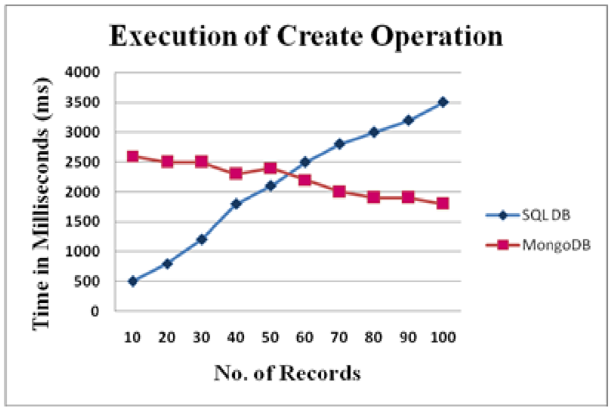


Fig. 9. Evaluated result of create operation.

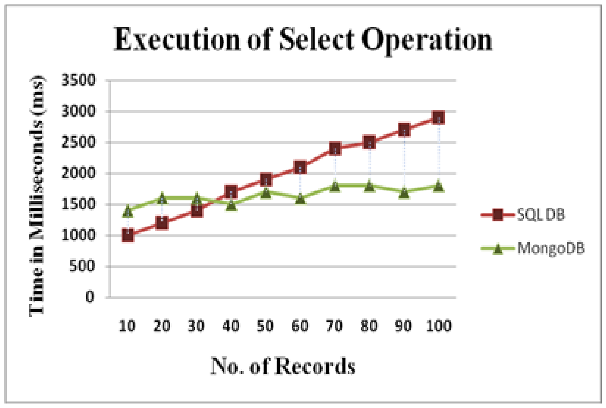


Fig. 10. Assessment map of select operation.

With respect to memory usage, Mongo DB preserves approximately 30% of memory as average when compared to MySQL. The efficiency mappings in association with basic queries into consideration are portrayed as Fig. 9 and 10, respectively.

V. Conclusion

It comes as no surprise that cloud and data storage provides enormous data storage that is compatible with ZB as well as mobile phones that provide storage in addition to 256 gigabytes (GB). A reliable, available, intact, quick, and secure database is necessary to extract intelligent information from this vast amount of data. The ideal substitute for handling massive data in multiple phases, such as transmission, storage, and analysis, is Mongo DB. Based on its schema information, an effort is performed here to convert the columnar SQL data to Mongo DB collections. The earlier methods mainly focused on relationship mode obtained by key and ignored the data volume and nature. Table type and size play important roles in selecting an appropriate transformation choice in this method.

The developed migration mechanism is tested using Python programming in the Mongo DB ATLAS environment. It captures the responsiveness in terms of transformation and query execution efficiency. A faster reaction is made possible by linking, which is 5.9% faster than two-way embedding, 7.6% faster than one-way embedding, and 8.4% faster than embedding. As comparison to the average relational table, the create operation performs 10% faster, and the select operation executes 13% faster. With relational databases taken into account, the memory utilization ratio of the migration procedure is estimated to be 30% on average.

In future the effectiveness of proposed migration model can be evaluated with complex queries including Update and Delete involving time-series data.

Authors' Contribution

Author 1 implemented the concept and drafted the article with assistance of authors 3 and 4, respectively. The author 2 reviewed the article.

Conflict of Interest

The authors declare that have no competing interest.

References

[1] 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.

[2] H. Alzahrani,, "Evolution of Object-Oriented Database Systems", Global Journal of Computer Science and Technology, vol. 16, no. 3, pp. 33-36, 2016.

[3] M. A. Ali, M. R. Ahmed, M. A. Khatun and K. Sundaraj, "A literature review on NoSQL database for big data processing", International Journal of Engineering & Technology, vol. 7, no. 2, pp. 902-906, 2018.

[4] J. R. Lourenco, B. Cabral, P. Carrerio, M. Vieira and J. Bernardino, "Choosing the right NoSQL database for the job: a quality attribute evaluation", Journal of Big Data, vol. 2, no. 18, pp. 1-26, 2015.

[5] Priyanka and Amit Pal, "A Review of NoSQL Databases, Types and Comparison with Relational Database", International Journal of Engineering Science and Computing, vol. 6, no. 5, pp. 4963-4966, 2016.

[6] V. Sharma and M. Dave, "SQL and NoSQL Databases", International Journal of Advanced Research in Computer Science and Software Engineering, vol. 2, no. 8, pp. 20-27, 2012.

[7] R. T. Mason, "NoSQL Databases and Data Modeling Techniques for a Document-oriented NoSQL Database", In Proc. of the Informing Science & IT Education Conference, pp. 259-268, 2015.

[8] D. G. Chandra, "BASE analysis of NoSQL database", Future Generation Computer Systems, vol. 52, pp. 13-21, 2015.

[9] K. Sahatqija, J. Ajdari, X. Zenuni, B. Raufi and F. Ismaili, "Comparison between relational and NOSQL databases", In Proc. of the International Convention MIPRO, pp. 216-221, 2018.

[10] S. Ghotiya, J. Mandal and S. Kandasamy, "Migration from relational to NoSQL database", In Proc. of IOP Conf. Series: Materials Science and Engineering, vol. 263, pp. 1-8, 2017.

[11] L. Stanescu, M. Brezovan and D. D. Burdescu, "An Algorithm for Mapping the Relational Databases To MongoDB – A Case Study", International Journal of Computer Science and Applications, vol. 14, no. 1, pp. 65-79, 2017.

[12] L. Stanescu, M. Brezovan and D. D. Burdescu, "Automatic Mapping of MySQL Databases to NoSQL MongoDB", In Proc. of the Federated Conference on Computer Science and Information Systems, vol. 8, pp. 837–840, 2016.

[13] L. Stanescu, M. Brezovan, C. A. Spahiu and D. D. Burdescu, "'A Framework for Mapping the MySQL Databases To MongoDB – Algorithm, Implementation and Experiments", International Journal of Computer Science and Applications, vol. 15, no. 1, pp. 65-82, 2018.

[14] A. A. Mahmood, "Automated Algorithm for Data Migration from Relational to NoSQL Databases", Al-Nahrain Journal for Engineering Sciences, vol. 21, no. 1, pp. 60-65, 2018.

[15] C. Gyorodi, R. Gyorodi, G. Pecherle and A. Olah, "A Comparative Study: MongoDB vs. MySQL", In Proc. of 13th International Conference on Engineering of Modern Electric Systems, pp. 1-6, 2015.

[16] L. Kumar, S. Rajawat and K. Joshi, “Comparative analysis of NoSQL (MongoDB) with MySQL Database”, International Journal of Modern Trends in Engineering and Research, vol. 2, no. 5, pp. 120-127, 2015.

[17] A. Krishan and R. Wagh, “A Study of Performance NoSQL Databases”, International Journal of Innovative Research in Advanced Engineering, vol. 4, no. 4, pp. 32-36, 2017.

[18] S. M. Nair, R. Roy and S. M. Varghese, “Performance Evaluation of MongoDB and CouchDB Databases”, International Journal of Scientific Research in Science and Technology, vol. 3, no. 7, pp. 21-24, 2017.

[19] R. S. Kumar and R. R. Mary, “Comparative Performance Analysis of various NoSQL Databases: MongoDB, Cassandra and HBase on Yahoo Cloud Server”, Imperial Journal of Interdisciplinary Research, vol. 3, no. 4, pp. 265-269, 2017.

[20] C. Chakraborttii, “Performance Evaluation of NoSQL Systems Using Yahoo Cloud Serving Benchmarking Tool”, In Proc. of UCSC Research Symposium, pp. 1-9, 2015.

[21] W. Saber, M. M. Eyada, M. M., El Genidy and F. Amer, “Performance Evaluation of IoT Data Management Using MongoDB Versus MySQL Databases in Different Cloud Environment”, IEEE Access, vol. 8, pp. 110656-110668, 2020.

[22] A. Singh, “Data Migration from Relational Database to MongoDB”, Global Journal of Computer Science and Technology, vol. 19, no. 2, pp. 17-21, 2019.

[23] I. S. Bajwa, S. Ramzan, B. Ramzan and W. Anwar, “Intelligent Data Engineering for Migration to NoSQL Based Secure Environments”, IEEE Access, vol. 7, pp. 69042-69057, 2019.

[24] R. S. Al-Mahruqi, M. H. Alalf and T. R. Dean, “A Semi-automated Framework for Migrating Web applications from SQL to Document Oriented NoSQL Database”, In Proc. of the 29th Annual International Conference on Computer Science and Software Engineering, pp. 1-10, 2019.

[25] M. P. Gopinath, G. S. Tamilzharasi, S. L. Aarthy and R. Mohanasundram, “An Analysis and Performance Evaluation of NOSQL Databases for Efficient Data Management in E-Health Clouds”, International Journal of Pure and Applied Mathematics, vol. 117, no. 21, pp. 177-197, 2017.

[26] F. Yassine and M. A. Awad, “Migrating from SQL to NOSQL Database: Practices and Analysis”, In Proc. of 13th International Conference on Innovations in Information Technology, pp. 58-62, 2018.