


# DLoader: Migration of Data from SQL to NoSQL Databases



Kanchana Rajaram , Pankaj Sharma, and S. Selvakumar

**Abstract** Data is increasing exponentially in the modern world which requires more proficiency from the available technologies of data storage and data processing. This continuous growth in the amount of structured, semi-structured, and unstructured data is called as big data. The storage and processing of big data through traditional relational database systems are not possible due to increased complexity and volume. Due to improved expertise of big data solutions in handling data, such as NoSQL caused the developers in the previous decade to start preferring big data databases, such as Apache Cassandra, MongoDB, and NoSQL. NoSQL is a modern database technology designed for fast read and write operations and provides horizontal scalability to store large amount of voluminous data. Large organizations face various challenges to shift their relational database framework to NoSQL database framework. In this paper, we proposed an approach to migrate the data from a relational database to the NoSQL database. We have specifically done transformation for Cassandra and MongoDB from MySQL database. The experiments show that the proposed approach successfully transforms the relational database to a big data database, and the performance analysis of such transformed databases shows that Cassandra database requires less storage space and offers a better performance.

**Keywords** Big data · Horizontal scalability · Relational database · Transformation · Cassandra · MongoDB

---

K. Rajaram (✉)

Department of Computer Science and Engineering, Sri Sivasubramaniya Nadar College of Engineering, Chennai, Tamil Nadu, India  
e-mail: [rkanch@ssn.edu.in](mailto:rkanch@ssn.edu.in)

P. Sharma · S. Selvakumar

IIIT Una, Una, Himachal Pradesh, India  
e-mail: [pankajk27897@gmail.com](mailto:pankajk27897@gmail.com)

S. Selvakumar

e-mail: [director@iiitu.ac.in](mailto:director@iiitu.ac.in)

## 1 Introduction

Relational databases have been the top choice of organizations in the last decade for storing, processing, and analyzing the data generated in the organizations. Relational databases store structured data and support structured query language (SQL) to access the database [1]. When the data is in structured format and of low volume, relational databases can be comfortably used. Even the unstructured or semi-structured data can also be stored in relational databases after using ETL tools to convert into structured format. However, recently, the data is increasing at an exponential rate having huge volume, high velocity of data generation, varied variety of the data as big data [2]. With this, it has become almost impossible for relational databases to store and process this huge amount of big data generated by large organizations and social media. Moreover, social networking and cloud computing paradigms have become popular which require data stores to manage massive amount of data generated per second. In particular, the cost of storing and querying big data from relational databases is very high, and they cannot serve the requirement of millions of users at the same time.

The big data storage and processing are not much suitable with relational databases as it has structured, semi-structured, and unstructured data, whereas relational databases can store only structured data and provide very little support to unstructured and semi-structured data [3]. Considering the challenges of big data for relational databases, the modern larger organizations which have large data storage requirements are rapidly switching from existing relational databases to NoSQL databases. NoSQL organizes data in different formats such as key-value, columnar, documents, and graphs. Cassandra stores data in columnar format, which offers considerably fast write operations [4], whereas MongoDB [5] stores data in document format. Every data type of NoSQL database uses different data structures as per the requirements.

The successful handling of big data complexity is a great challenge for the conversion of relational databases to NoSQL databases. NoSQL databases are designed according to the application specific access patterns and queries without using a normalization process, and they do not support any join operation and foreign or primary key relations. It is challenging for the organizations to migrate data from the relation databases to NoSQL databases because they completely work on different technologies. To overcome this problem, we have proposed a framework, namely DLoader, to migrate data from relational database such as MySQL database to NoSQL databases such as Cassandra and MongoDB. The proposed approach DLoader involves extraction of data from the MySQL database, preprocessing the data by applying transformations and standardizations on the data, and finally mapping the columns in the MySQL tables into the fields of NoSQL databases.

The rest of the paper is structured as follows: Review of the related work is presented in Sects. 2 and 3 describes the proposed work; Sect. 4 describes about the experimentation with Cassandra and MongoDB. Lastly, Sect. 5 concludes the work and suggests a future work.

## 2 Literature Review

NoSQL is a modern database technology to store and process big data. Most of the utilities developed were developed for converting one form of SQL database to another SQL database, and no specific utility is available for NoSQL to NoSQL conversion. A data adapter system was proposed to promote hybrid database architecture including both SQL and NoSQL [6]. Many frameworks have been developed to solve the data migration problem with a different solution, characteristics, and properties which migrates data from one NoSQL database to another but not between SQL to NoSQL [7]. It provided algorithm migration and migration schemes to migrate data between NoSQL databases in actual operating environment. It is a challenging task for business organizations to migrate or transform their data from existing relational database to NoSQL databases due to the complexity in relational data.

An approach consists of two modules: Data transformation and data cleansing modules were proposed which transforms relational database to big data database [8]. To handle the complexity of automatic transformation of existing relational database into a NoSQL database, a bi-fold transformation consisting of schema-to-schema and data-to-data transformation approach was proposed which dealt with heterogeneous and complex data [9]. Heterogeneous data exchange and conversion of data across kinds of database systems are achieved through relationship schema mapping [10]. It specifically focused on exchanging XML heterogeneous, solving field attribute changing during migration problem.

A study revealed that NoSQL is faster than RDBMS in case of big data. A methodology was presented for data migration from MySQL to MongoDB as NoSQL database [11]. A study evaluated the performance of data insertion and retrieval speeds up by making comparison between MongoDB as NoSQL and MySQL as a relational database which showed NoSQL database is faster than the MySQL considering the parameters used in the study for huge amount of data [12]. An approach that used data and query features to migrate data from relational to NoSQL databases preserved the features in the source data in the relational database and queries for accessing the data on the source data as well for arranging the data in the target database [13, 14]. This system works for all NoSQL databases but requires meta data information of the source and target databases.

NoSQL layer can also be used as an interface resides between the source database and application to migrate the data between relational to NoSQL without changing the application code which gives high performance for MongoDB [15]. Other approaches include data adapter approach which integrates relational database and NoSQL databases which support hybrid database architecture. Data may not be consistent in this approach always [16]. Content management system for schema de-normalization works with the Hadoop framework [17]. In a study, document-oriented data schema was proposed covering all data types in databases, further it overcomes the issue of managing the relationships of a complex database. Two stages of the approach include designing the document-oriented data schema and migrating the ER model to the document-oriented data schema [18]. Data migrated from relational to NoSQL

data models or between different NoSQL platforms needed to be validated in order to find the errors during transformation during migration of the data. Data can be validated using denormalized schema structures and bloom filters, and other approaches can also be implemented to validate the data [19].

### 3 Proposed Work

The concepts of relational databases and NoSQL databases and the differences between these two databases are discussed in the next two sub-sections. Thereafter, the proposed approach for transforming the data from relational databases to NoSQL databases is elaborated.

#### 3.1 *Relational Databases*

Relational or structural databases are based on relational model. Data stored in these tables is related and can be accessed through the relationships between them. The relationships can be one-to-one, one-to-many, and many-to-many. Data stored in the relational databases is structured only. Relational databases consist of tables, and the tables consist of rows and columns. There can be number of rows in a table, and all the rows are of same type that contains different data. Relational database management system (RDBMS) is used for creation and management of the relational databases. Each row in a table is unique. There are mainly two types of keys in RDBMS: primary key and foreign key. Primary key consists of one or more columns which can uniquely identify all the records in a table, and foreign key consists of one or more columns of a table that refers to the primary key of another table. Relational databases also follow ACID properties (atomicity, consistency, isolation, and durability) for making transactions. For querying the data from the relational models, structured query language (SQL) is used. These databases should have a predefined schema which is not easy to alter if structure of the data coming into the database changes with time. Hence, schema must be strictly designed keeping the future requirements in the mind.

#### 3.2 *NoSQL Databases*

NoSQL databases have been designed to facilitate big data storage and processing. NoSQL databases store data in various formats such as key-value, column store, documents, and graphs. They have their own query mechanism and do not have any specific query language. Key-value databases consist of items where each item contains keys and values. A value can be simply retrieved by referring its key. Columnar databases

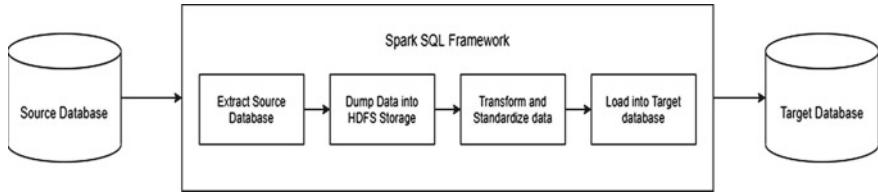


Fig. 1 SQL to NoSQL data migration

consist of tables which consist of rows and dynamic columns. They have flexible schema and number of columns can be increased with time without creating a new schema. The column names need not be predefined in NoSQL databases, and hence, the structure is not fixed. Columns in a row are kept in a sorted order according to their keys which include partition keys and clustering keys. Document type NoSQL databases store data in documents which is same as JSON objects. The collection of documents is called a collection. Each document contains key-value pairs where value can be of different types. Graph databases store data in nodes and edges. Nodes generally hold information about entities, whereas edges store information about the relationship between the nodes. NoSQL databases have denormalized structure as they do not have primary and foreign key concept and hence do not support join operations. They provide features like horizontal scalability, data replication, data availability, and data consistency at low cost. With the increasing amount of data, the NoSQL database can be easily scaled, and more data storage can be accommodated [20].

### 3.3 SQL to NoSQL Transformation

Data stored in the relational database in the tables must be mapped to NoSQL database tables column to column. We have used Spark framework for migrating data from SQL to NoSQL database [21]. Hadoop cluster is serves as a middle layer between the source and target database. NoSQL database such as Cassandra also supports collection data types such as set, map, and list, and tables are denormalized, so columns in the SQL database can also be mapped to collection data types. The detailed process is briefed in the further sub-sections. The steps in migrating the data from a relational database to a NoSQL database are shown in Fig. 1.

#### 3.3.1 Data Loading

Data in the relational database stored in MySQL holds a specific schema in which various tables are linked through primary and foreign key references. Each column in a table has a defined data type. We have made use of timestamps to get the data

from the MySQL database. Timestamp is used to make sure that only recently added data gets retrieved. JDBC is used to establish connection with the source database. The data from the MySQL database is loaded into the Hadoop distributed file storage (HDFS) [22] using Spark SQL framework. HDFS is a clustered storage consisting of one name node and N data nodes. This Hadoop cluster acts as a middle layer between the source database (MySQL) and the target database (NoSQL). SQL like queries can also be made on the data in the Spark data frames by creating temporary tables by using Spark SQL. Spark SQL also supports join and aggregate functions as supported in the SQL. The incoming data from the source database is dumped in the HDFS keeping the schema, the same as that of source database. Data is temporarily stored in the HDFS as staging tables before applying processing the source data.

### 3.3.2 Data Transformation and Mapping

HDFS holds large amount of data dumped from the relational databases (MySQL) in delta file format without altering the schema of the source database. This data is loaded in a data frame of the Spark SQL framework, one table at a time. Transformations are applied on the table data contained in the data frame, keeping the target database schema in mind. Since the target NoSQL database does not support joins, the MySQL source tables are denormalized to enable migration of data from the relational database. Every column of a table in the source database is transformed according to the column schema of the target database. Different concepts in relational databases are mapped to NoSQL databases as shown in Fig. 2.

Transformation includes changing columns names, changing data type of columns according to target database, altering the tables by adding, or removing the columns according to the target database schema. Standardization includes ensuring consistent date formats, uniform format for name values, uniform abbreviations, for example, for gender column value, consistent format for mobile number values, etc. Finally, the transformed data frame is loaded into a particular table of target NoSQL database in append mode using Spark SQL. The algorithm for preprocessing and mapping the source data to target is given below:

#### Algorithm 1 Data preprocessing and mapping

Algorithm Data preprocessing mapping

Input: Relational database

Output: NoSQL database

1. Establish a connection with source database.
2. Create table object for source.
3. Dump the table objects in the HDFS storage.
4. Load the data from all the tables into HDFS storage
5. For each column of each of the tables, if transformation rule is applicable: Transform the columns according to the target table columns.
6. Standardize the content of the columns.
7. Map the HDFS table objects into target table objects.
8. Map the HDFS column objects into target column objects.

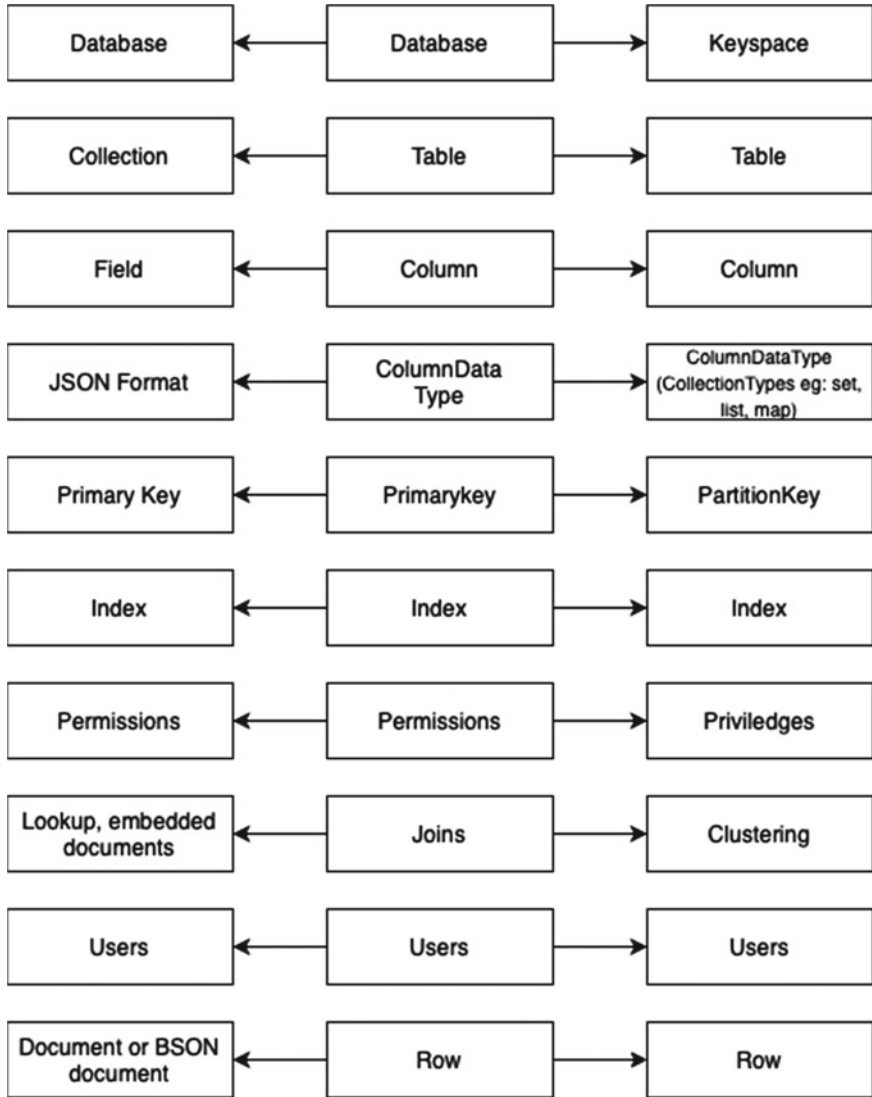


Fig. 2 Concept mapping between relational database and NoSQL database

- 9. Construct each target table.
- 10. Construct target object of each column.
- 11. Select partition key and clustering key for each target table..
- 12. Compare partition key of the target tables to keep track of the duplicate records.

## 4 Experimentation

For experimentation purpose, a test bed comprising of a HDFS-based cluster with a name node and three data nodes has been set up. The name node is a Intel Xeon server 3.3 GHz, 32 GB RAM, 4 Cores with 2 TB HDD, and the data nodes are Intel i-4 core workstations with 16 GB RAM and 1 TB HDD. Our own dataset has been generated consisting of immunization data related to children of Tamil Nadu state. The average population of Tamil Nadu state is 7.7 crore in 2020 [23], and around 30% of them are children. The number of children approximately in a state could be 2.3 crores. The immunization database consists of tables for storing details of children, their mothers, and immunization details. The data is generated using a tool called DbGen [24] and using MS Excel. DbGen is a Windows-based tool that can be configured based on the schema to generate data.

The immunization data in the MySQL database is migrated using our proposed approach of DLoader into two NoSQL databases such as Cassandra and MongoDB. For a given number of children records, the associated mother records and immunization records are also considered to be migrated from the source database. When loading the same number of records, the database sizes of Cassandra and MongoDB are different. The number of records used in the dataset along with the varied size of Cassandra and MongoDB databases are shown in Table 1. The number of children records has been varied from 10 lakh to 2.5 crores. It is observed that Cassandra offers good compression of data and stores the same number of records in the source database in storage space of 41% lesser when compared to MongoDB.

As a second experiment, the performance of both NoSQL databases is analyzed in terms of response time and throughput. The load testing has been done using Jmeter tool [25]. The following two queries have been considered.

Q1. Retrieve the details of a particular child whose *child\_id* is given.

**Table 1** Varied sizes of NoSQL databases for storing given relational data

No. of children	Cassandra DB				MongoDB			
	Child master DB Size (GB)	Mother master DB size (GB)	Imm. DB size (GB)	Total DB size (GB)	Child master DB Size (GB)	Mother master DB size (GB)	Imm. DB size (GB)	Total DB size (GB)
1,000,000	0.07	0.1	0.1	0.3	0.07	0.2	0.2	0.5
2,000,000	0.1	0.3	0.1	0.7	0.1	0.4	0.3	0.9
3,000,000	0.2	0.5	0.2	1.0	0.2	0.7	0.7	1.3
5,000,000	0.4	0.9	0.4	1.5	0.3	1.0	1.0	2.1
10,000,000	0.9	1.8	0.8	3.5	0.7	2.0	2.0	4.7
15,000,000	1.2	2.7	1.2	5.0	1.1	3.0	3.0	7.1
20,000,000	1.6	3.6	1.8	6.8	1.3	4.0	4.0	9.3
25,000,000	2.1	4.5	2.0	8.0	1.8	5.0	5.0	11.8



**Table 2** Performance analysis of Cassandra and MongoDB

S. No.	Cassandra DB					MongoDB				
	DB size (GB)	Throughput		Response time (s)		DB size (GB)	Throughput		Response time (s)	
		Q1	Q2	Q1	Q2		Q1	Q2	Q1	Q2
1	0.3	0.44	0.34	2.29	2.95	0.5	0.39	0.17	2.54	5.98
2	0.7	0.44	0.33	3.06	3.01	0.9	0.33	0.15	3.09	6.60
3	1.0	0.43	0.33	3.40	4.42	1.3	0.31	0.14	4.01	9.57
4	1.5	0.33	0.32	4.08	5.01	2.1	0.29	0.14	4.80	11.02
5	3.5	0.29	0.26	8.07	10.05	4.7	0.22	0.10	9.32	20.04
6	5.0	0.21	0.19	10.80	18.02	7.1	0.19	0.06	13.50	32.00
7	6.8	0.15	0.12	15.00	22.05	9.3	0.12	0.03	20.04	40.30
8	8.0	0.10	0.08	20.97	26.45	11.8	0.09	0.02	30.50	50.02

Q2. Retrieve immunization details of a children in a particular routine immunization session.

The data load is varied from 0.3 to 8 GB of source MySQL database and migrated to Cassandra as well as MongoDB. For each of the queries Q1 and Q2, the throughput and response time are computed and shown in Table 2. The queries were executed three times under different network loads. The average throughput and maximum response time in seconds for Cassandra and MongoDB databases among three runs are shown in the table.

The performance of Cassandra and MongoDB by varying the data load for two different queries is shown in Fig. 3. The following observations are made from this experiment.

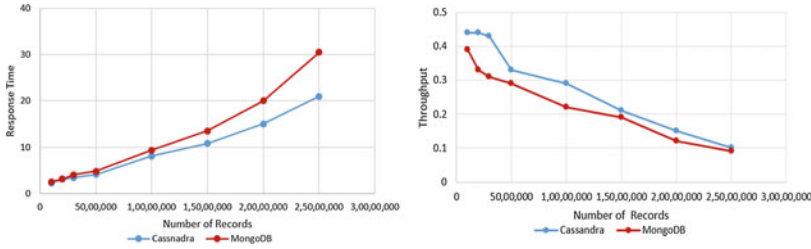
For query Q1

- For 26-fold increase in Cassandra database size, there is only ninefold increase in Response time and throughput decreases by 77%.
- For 26-fold increase in MongoDB size, there is only 12-fold increase in response time and throughput decreases by 77%.

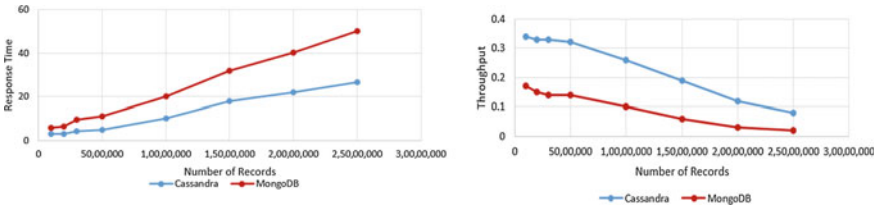
For query Q2

- For 26-fold increase in Cassandra database size, there is 12-fold increase in response time and throughput decreases by 76%.
- For 26-fold increase in MongoDB size, there is eightfold increase in response time and throughput decreases by 88%.

The rate of increase in response time is more in case of Cassandra for both the queries when compared to MongoDB. However, in Cassandra, the response time for queries Q1 and Q2 for any database size is lesser by 16% and 50%, respectively, as compared to MongoDB.



(a) For Query 1



(b) For Query 2

Fig. 3 Performance analysis of Cassandra and MongoDB

### 5 Conclusion

The proposed system DLoader for migration of data from SQL to NoSQL database is the generalized system which is capable to migrating any schema in SQL to NoSQL database. Spark framework which provides fast performance through its inbuilt parallel processing is used for data extraction, transformation, standardization, and data loading. Spark cluster-based HDFS storage is used in the middle layer to handle large amount of data at a time and makes the data readily available during the transformation and mapping process. The proposed approach has achieved column to column mapping from source to target database. The proposed approach is tested for migrating relational data to NoSQL databases such as Cassandra and MongoDB. Cassandra consumes less storage space and offers better performance as compared to MongoDB.

**Acknowledgements** This work was supported by Grand Challenges India (GCI) for Immunization Data: Innovating for Action (IDIA) funded by BIRAC and jointly funded by Department of Biotechnology and Bill & Melinda Gates foundation.

## References

1. S. Ramzan, I.S. Bajwa, B. Ramzan, W. Anwar, Intelligent data engineering for migration to NoSQL based secure environments. *IEEE Access* **7**, 69042–69057 (2019)
2. A. Katal, M. Wazid, R.H. Goudar, Big data: issues, challenges, tools and good practices, in *2013 Sixth international conference on contemporary computing (IC3)* (IEEE, 2013), pp. 404–409
3. M. Potey, M. Digrase, G. Deshmukh, M. Nerkar, Database migration from structured database to non-structured database. *Int. J. Comput. Appl.* **975**, 8887 (2015)
4. V.D. Jogi, A. Sinha, Performance evaluation of MySQL, Cassandra and HBase for heavy write operation, in *2016 3rd International Conference on Recent Advances in Information Technology (RAIT)* (IEEE, 2016), pp. 586–590
5. MongoDB (2021) <https://www.mongodb.com/>
6. S. Ghule, R. Vadali, Transformation of SQL system to NoSQL system and performing data analytics using SVM, in *2017 International Conference on Trends in Electronics and Informatics (ICEI)* (IEEE, 2017), pp. 883–887
7. Y.S. Wijaya, A.A. Arman, A framework for data migration between different data store of NoSQL database, in *2018 International Conference on ICT for Smart Society (ICISS)* (IEEE, 2018), pp. 1–6
8. A.E. Lotfy, A.I. Saleh, H.A. El-Ghareeb, H.A. Ali, A middle layer solution to support ACID properties for NoSQL databases. *J. King Saud Univ.-Comput. Inf. Sci.* **28**(1), 133–145 (2016)
9. S. Ramzan, I.S. Bajwa, R. Kazmi, An intelligent approach for handling complexity by migrating from conventional databases to big data. *Symmetry* **10**(12), 698 (2018)
10. N. Li, B. Xu, X. Zhao, Z. Deng, Database conversion based on relationship schema mapping, in *2011 International Conference on Internet Technology and Applications* (IEEE, 2011), pp. 1–5
11. M. Hanine, A. Bendarag, O. Boutkhom, Data migration methodology from relational to NoSQL databases. *World Acad. Sci. Eng. Technol. Int. J. Comput. Electr. Autom. Control Inf. Eng.* **9**(12), 2369–2373 (2016)
12. A. Abdullah, Q. Zhuge, From relational databases to NoSQL databases: performance evaluation. *Res. J. Appl. Sci. Eng. Technol.* **11**(4), 434–439 (2015)
13. S. Ghotiya, J. Mandal, S. Kandasamy, Migration from relational to NoSQL database, in *IOP Conference Series: Materials Science and Engineering*, vol. 263, no. 4 (IOP Publishing, 2017), p. 042055
14. D. Liang, Y. Lin, G. Ding, Mid-model design used in model transition and data migration between relational databases and nosql databases, in *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, (IEEE, 2015), pp. 866–869
15. L. Rocha, F. Vale, E. Cirilo, D. Barbosa, F. Mourão, A framework for migrating relational datasets to NoSQL. *Procedia Comput. Sci.* **51**, 2593–2602 (2015)
16. Y.T. Liao, J. Zhou, C.H. Lu, S.C. Chen, C.H. Hsu, W. Chen, M.F. Jiang, Y.C. Chung, Data adapter for querying and transformation between SQL and NoSQL database. *Future Gener. Comput. Syst.* **65**, 111–121 (2016)
17. C.H. Lee, Y.L. Zheng, SQL-to-NoSQL schema denormalization and migration: a study on content management systems, in *2015 IEEE International Conference on Systems, Man, and Cybernetics*, (IEEE, 2015), pp. 2022–2026
18. S. Hamouda, Z. Zainol, Document-oriented data schema for relational database migration to NoSQL, in *2017 International Conference on Big Data Innovations and Applications (Innovate-data)* (IEEE, 2017), pp. 43–50
19. A. Goyal, A. Swaminathan, R. Pande, V. Attar, Cross platform (RDBMS to NoSQL) database validation tool using bloom filter, in *2016 International Conference on Recent Trends in Information Technology (ICRTIT)* (IEEE, 2016), pp. 1–5
20. Y. Huang, T.J. Luo, Nosql database: a scalable, availability, high performance storage for big data, in *Joint International Conference on Pervasive Computing and the Networked World* (Springer, Cham, 2013), pp. 172–183
21. K. Atkotiya, P. Shukla, Migration from relational database like MySQL to nosql database like Cassandra is necessary and how to migrate it using spark

22. Apache Hadoop (2021) <http://hadoop.apache.org/>
23. Rural Health Statistics <https://hmis.nhp.gov.in/downloadfile?filepath=publications/Rural-Health-statistics/RHS%202019-20.pdf>
24. DbGen Database Management Tool (2021) <https://www.bcdsoftware.com/series400solutions/dbgen/>
25. Apache Jmeter (2021) <https://jmeter.apache.org>