International Journal of Informatics and Applied Mathematics
e-ISSN:2667-6990 Vol. 3, No. 2, 53-65

An Overview on the Mapping Techniques in NoSQL Databases

Aicha AGGOUNE

LabSTIC Laboratory, University of 08th may 1945, Guelma, Algeria
aggoune.aicha@univ-guelma.dz

Abstract. In the last decade, the data mapping of NoSQL stores and the ability to perform massively parallel data processing have drawn much interest from researchers. The data mapping techniques have been mainly developed for relational database systems in centralized environments. However, the NoSQL approach is not the same as relational one. It aims to handle a large volume and high speed of data with various structures in the distributed environment. Adapting the data mapping techniques to various NoSQL types improves the performance and data storage in the suitable NoSQL database. This paper investigates the recent techniques of mapping between NoSQL databases and discusses their advantages as well as their disadvantages.

Keywords: Data Conversion · Data Mapping · NoSQL Approach · NoSQL Models.

1 Introduction

The usage of NoSQL databases becomes more and more increased in different Internet applications for example, social networking sites, cloud computing, commercial sites such as Amazon, Fnac, etc. These databases enable massive data volumes, so-called big data to be stored and processed. The storage and processing of these huge amounts of data can be done by data partitioning across various nodes in the distributed environment [19]. The distributed relational databases provide a high consistency in which the data values are the same for all nodes [28]. However, many issues have appeared, including the complexity of handling distributed queries over a huge amount of data, the availability of these data volumes on the network cannot be ensured, and the scalability of adding nodes is not easy [43]. The relational database management system (RDBMS) is unable to store and process a large amount of data in a distributed environment [28]. In this context, a new approach for big data modeling has emerged, so-called NoSQL (Not Only SQL).

NoSQL approach was introduced to deal with the problems of big data, for providing highly scalable systems that are also high performance and highly available [29]. Unlike to relational database, the data stored in NoSQL are mostly unstructured or semi-structured [34]. There is no fixed schema of the NoSQL database, which must be followed like in the relational databases. The storage of NoSQL data is made in a very exible way without any constraints or any designed data schema. Hence, the NoSQL system allows the modification of data structure during database usage whenever required due to the schema-free nature of data, so-called schema-less [34]. Also, the NoSQL data can be modeled in different data models including key-value model, columnar-oriented model, document-oriented model, and graph-oriented model [38]. Therefore we can build four different NoSQL databases with the same data, for instance document-oriented database using MongoDB system and columnar-oriented database by using Cassandra system. Some NoSQL databases may be more fitting than others for the representation and management of the data that needs to be used. The need to use the suitable NoSQL database was in fact, motivated by the need to map between different NoSQL databases. The main goal of this paper is to review current approaches and frameworks for mapping between NoSQL databases.

The rest of the paper is organized as follows. In Section 2 , we present an overview of NoSQL data. In Section 3, we illustrate the features of the principal models of NoSQL databases. In Section 4, we review related work on mapping between NoSQL databases with a comparison study between frameworks in section 5. Finally, a conclusion and perspectives are given in Section 6.

2 Overview of NoSQL Data

Due to the important development in computer Hardware, intelligent systems, sensors, and internet-connected devices, the data become more widely used anytime. Hence, the data sources become so large and contain various data types, which are continuously, and rapidly developed. NoSQL data represent today an emerging concept in the data science domain for collecting, storing, organizing, analyzing, and processing large amounts of data, so-called big data [20]. NoSQL data referred to as Not Only SQL, tended to store and maintain massive data that could not be managed by relational database management systems (RDBMS) [46]. The term not only SQL means that the large scale of data can accommodate a wide variety of data models, not only the relational model which uses SQL as a query language, but also other models, such as key-value, document, column, and graph formats [21].

In particular there are significantly more studies on the comparison between SQL and NoSQL databases features [31, 35]. In Table 1, we briefly give the principal differences in features between NoSQL and SQL databases.

Table 1. NoSQL versus SQL databases.

Features	NoSQL database	SQL database
Data size	Large-scale data	Size $\leqslant$ 128 terabytes
Data structure	Unstructured or semi-structured data	Structured data (tables)
Data type	Various data type	Homogeneous data type
Scalability	Horizontal scalable	Vertical scalable
Availability	High available	Available with high cost
Consistency	Difficult to achieve	High consistent
Partition tolerance	High tolerant	Less partition tolerant
Query complexity	Simple queries	More complex
Query language	No standard language	SQL language
Data schema	Schema less	Rigid schema
Data model	Different data models	Relational model
Data security	Difcult to achieve	Easy to achieve

NoSQL systems manage their data over a distributed architecture in a cluster of servers or over a grid in different geographical locations or the cloud computing environment [38]. They ensure high performance in which the data are horizontal scaled by adding new machines and high availability due to the simpler data model but they do not guarantee data consistency at all times. Thus, the NoSQL database works well despite network and node failures [22]. The data stored in the NoSQL databases can be represented in different structures without requiring a fixed schema for tables, unlike in the RDBMS. In contrast to the SQL database that is based on the relational model, NoSQL data can be modeled in different data models such as Key-value, Document, Columnar, and Graph [34].

3 Description of NoSQL Data Models

NoSQL system reveals high exibility and availability of a huge amount of data, thereby, it results in high performance. NoSQL data have received recently a high level of attention, and their usage is still growing. The important large companies are taking on NoSQL-based data stores such as Google, Facebook, Twitter, Amazon, eBay and many other. With the heterogeneity of the large amount of data, the NoSQL systems organize their data according to dierent data models, which are classified into four types, namely [22, 30, 34]:

– **Key-value model:** it represents the data in the simplest form by using a key for uniquely identifying the diverse data. Each key is linked with its data that may be either a simple value, such as a number, or a complex value, for instance, a document. The querying of key-value databases can achieve low latency as well as high performance. However, if a system claims more complex operations, this data model is not powerful enough. The key-value model is extremely useful for log files management and caching in web applications, storing and processing session data, which include a unique key for each user session and a value that represents the user profile information, messages, etc. An example of key-value systems: Redis, Riak, OracleNoSQL, and Memcached [12, 14–16].
– **Document-oriented model:** it is a special case of the key-value model, in which the value is limited to documents in various formats (JSON, BSON, XML, etc.). It supports efcient querying compared to the key-value model, and it provides more efficient data access to aggregates. The document-oriented model is used to represent the documents on the web, content management, and mobile application data processing. It is widely employed to store data by using the popular NoSQL DBMS, namely MongoDB [13]. Other document-oriented systems are CouchDB, MarkLogic, DocumentDB, etc. [5, 7, 11].
– **Wide columnar-oriented model:** in this model, the data are represented by a row key as an identifier, along with a series of columns representing the same subject that forms the column family. The different column families can be distributed over various nodes. This model is commonly used to store and process a large number of tables with high partition data over several nodes. It is more suitable for accounting, averaging, and stock management. The wide columnar databases do not aggregate all data from each row, but instead values of the same column family and from the same row. The most used wide columnar-oriented systems are Cassandra, HBASE, and Azure Table [3, 4, 8].
– **Graph-oriented model:** The data represented by a graph, where the nodes represent values and the edges describe relationships between different nodes. The graph-oriented stores are widely used in social, biological network, electr- onic system data, and services to establish relationships and properties betw- een multiple values. HypergraphDB, AllegroGraph, IBM Graph, and Neo4J are four examples of a graph-oriented system [1, 9, 10, 44]. The Neo4j is the popular graph-oriented system because it offers so-called ACID transaction properties (atomic, consistency, isolation, and durability). [44].

Figure 1, illustrates the difference categories of NoSQL databases.

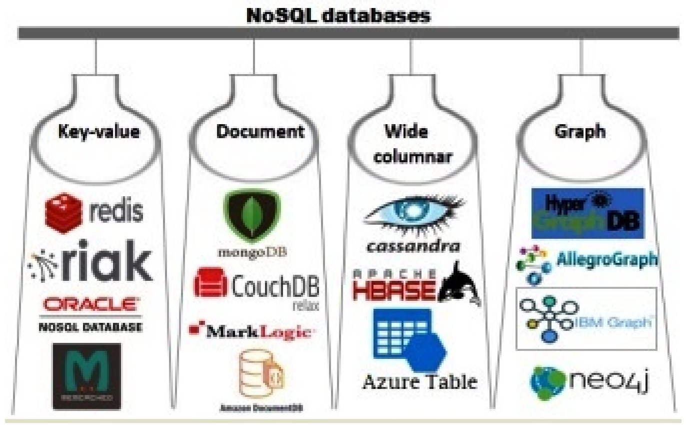

Fig. 1. Some NoSQL databases of each data model.

There are several NoSQL database management systems (NoSQL DBMS) of each NoSQL data model, which differ in their features and properties, for example API operations, query languages and programming language (PL), etc. [33]. Some NoSQL DBMS have multi-model databases, including Apache's ArangoDB, which can manage each of the Key-value, Document, and graph stores. It combines the flexibility of JSON with semantic search and graph technology for next-generation feature extraction even for large datasets [2].Table 2 shows the important features of each NoSQL data models.

Table 2. Description of NoSQL data models.

Data model	Performance	Querying	Utility	Usability
Key-value	Simplest structure Data share	Access data by key value only	More efficient to represent simple data sets	Restrictive for a simpler data with a large volume
Document	Data persistance More flexible with sharding	Content-based querying	More efficient to manage big data applications	Store large scale documents
Wide columnar	Data persistance Data compression	Consistent data access to big table	Avoids Input/ output overhead	Store very large tables
Graph	Highly connected data with low latency	Querying multiple relationships inside large data sets	More suitable managing the linked data	Store the wide linked data

Data mapping from one database to another is very important with the heterogeneity of the NoSQL data model as well as the variety of NoSQL DBMS.

4 NoSQL Data Mapping Approaches

The diversity of NoSQL data models has allowed us to ask two important research questions:

i What is the appropriate NoSQL data model for the storage and management of data in a given field?
ii How to map the existing NoSQL database to the best NoSQL category?

The answers to these questions open up a new opportunity to explore the different NoSQL databases and suggest mapping solutions between them. Due to the presence of more than 225 NoSQL databases [41], we present at the first stage comparison between two widely used databases, which are: MongoDB and Cassandra NoSQL databases.

4.1 Comparison between MongoDB and Cassandra

We review the different characteristics and features of the two most popular NoSQL databases for storing and managing large amounts of data: MongoDB as a document-oriented database and Cassandra as a wide columnar database.

MongoDB is a document-oriented database management system that was initially released in 2009 [13]. It was developed by MongoDB, Inc (formerly 10gen) using C++ as a programming language. MongoDB stores the data as a document in binary encoded JSON format (BSON). The BSON document contains an ordered list of pairs (field name, value) where the value may be simple or complex, such as an embedded document, a referenced document, a list, or array. MongoDB is currently being used by MTV networks, GitHub, and Foursquare.

Data querying is accomplished by Mongo shell commands that are represented in JSON syntax and sent to MongoDB as BSON objects via the database driver. In addition, MongoDB is based on the MapReduce paradigm to more express complex queries.

There are several important features for the high performance and efficient MongoDB database, namely: complex aggregation with the use of MapReduce for statistical analysis, indexing over embedded objects, replication to improve data readability, and the automatic sharding into shards (partitions or chinks) to ensure the high scalability of a distributed data [27].

The data in MongoDB database can be managed under Master-Slave architecture with high availability by the support for replication [27]. The replica pair architecture provides a high partition tolerance when a slave or current master of the replica pair fails.

Cassandra is an open source, wide columnar database, which was originally developed at Facebook in 2008 and it has been an Apache Software Foundation, Top level project (TLP) since 2009 [4]. Cassandra is written in JAVA and uses the Thrift API for data access. Cassandra is a leading transactio- nal distributed database used by Facebook for handling a large amount of data across many commodity servers. Cassandra was used by other major companies such as IBM, Twitter, Rackspace, etc. Cassandra data storage is very similar to relational databases, generated tables, columns, and rows, but does not support inter-table joint operations. Unlike the relational model, Cassandra data are arranged in columns rather than rows, where these columns do not have to be the same in every row. Cassandra maps a tuple consisting of a row key, a column name, and a timestamp to a value. Columns are organized into column families, which are similar to the relational model tables.

Cassandras data model is based on two NoSQL models, the wide columnar model of Google Big Table and the key-value model of Amazon Dynamo [32]. Further, unlike the MongoDB, the Cassandra database uses peer-to-peer replication, ensuring high availability, scalability with high partition tolerance and persistence. The data are distributed and replicated across nodes, and all clusters have equal permissions [32]. Cassandra provides two replication strategies: a simple strategy used when the cluster is deployed across one datacenter (a group of related nodes), and the network topology strategy, which is used when the cluster is deployed across multiple datacenters.

Apache Cassandra has its own query language called Cassandra Query Language (CQL), which is similar to SQL. CQLSH is a command line shell for interacting with Cassandra through CQL. DataStax entreprise (DSE), a software publisher that provides Apache Cassandra with additional tools and functionality for analysis, research, security, monitoring and graphical administration, as well as the ability to work in memory.

Much work has been done on a comparative study of the most important NoSQL databases [27,32] and according to the details given by DB-engines.com website [6], we summarize in Table 3 the main differences between MongoDB and Cassandra databases.

Table 3. Main differences between MongoDB and Cassandra.

Criteria	MongoDB	Cassandra
Data model	Document	Wide Columnar
Implementation language	C++	Java
Developer	MongoDB, Inc	Originally developed by Facebook then Apache TLP
Query language	Mongo shell	CQL
Cloud database service	MongoDB Atlas, ScaleGrid for MongoDB	Cassandra datastax
Data access API	Proprietary protocol using JSON	Thrift
Replication	Master-Slave	Peer to Peer
Data storage	Multiple data storage both in memory and on disk	Data storage on disk only

4.2 Mapping Approaches

There are several reasons for mapping between NoSQL databases, which we list six good reasons:

1. Some NoSQL databases can be better suited than others to represent the data to be used.
2. The appropriate NoSQL database can enhance the executing low-latency queries and reduce the cost against other databases.
3. Displeasure in terms of data querying and query optimization in the old database.
4. The need to provide data portability between different NoSQL databases.
5. The need for a better solution to use business or technology strategy in a good case means thinking about mapping data rather than building a new database from scratch.
6. Data mapping is more convenient for incorporating heterogeneous NoSQL databases into data integration systems such as the mediator system and Datawarehouse.

Shirazi et al. [39] have proposed a design pattern-based approach for bidirectional mapping between column-oriented database and graph-oriented database. The design pattern is a solution to common problems in software design. This data mapping approach offers cloud data portability, which enables data and applications to be moved from one cloud provider to another. The principle idea of this approach is to define two design patterns. The first one aims to ensure the mapping of column-oriented to graph-oriented database. The second one provides data mapping in the other direction.

The approach has been applied in the healthcare domain, and the results show that the graph-oriented database is more suitable for representing the complexity data than the column-oriented database, which is more convenient for maintaining the large amount of data. This approach provides a better result if the graph-oriented database has to be transformed into a column-oriented database. The mapping in the other direction however poses some problems with the data quantity.

Scavuzzo et al [36] have proposed a meta-model approach for mapping between two specific columnar databases, which are Google App Engine Datastore, and Microsoft Windows Azure Tables. The meta-model is an intermediate model between the source and the target models. This approach used BigTable data model as a support for columnar databases. The proposed approach is focused on a number of extractors, translators and reverse translators, by extracting data from the source database and transmitting them to translators that in turn translate it into a meta-model format. The reverse translators use the transformed data as input and provide data as output in a target format.

This approach has evaluated using data from meeting in the cloud and the results show that the extraction and conversion time is less than 0.1% of the time needed for the complete mapping. Despite this approach allows adding new data, but it requires significant mapping time and does not guarantee that all parts of the data have been translated.

In 2016, Scavuzzo et al [37] have enhanced their previous work, by supporting fault tolerance in the mapping of huge amounts of columnar databases. The extended approach offers a virtual data partitions (VDPs) of the source database to provide a set of parallel mappings of VDP instead mapping the entire database. This approach is more efficient than the previous one with a speedup of 25 times without losing data.

Thalheim and Wang [42] proposed an approach derived from the data warehouse technique, well known by ETL (Extract, Transform and Load), and applied general refinement theory for data mapping. Data sources move from legacy systems into new system in which data sources have different structures. The refinement theory specified two sub-classes of transformations: property-preserving and property-enhancing transformations.

Bansel et al. [26] proposed an approach based on an online compression algorithm for mapping between document and graph databases in the cloud environment. This mapping can be achieved directly or indirectly through the intermediate model. Indirect mapping consists of transforming document databases to columnar and to transform them into the graph format. The experimental study is based on the use of two JSON data sets: Core US Fundamental for nance and economic data, and Twitter tweets from November 2012. The results show that the intermediate mapping enhances the read/write efciency.

Androec and Vrek [25] proposed a semantic web services-based approach for the data mapping between different cloud storage systems. The authors focused on dealing with the data lock-in problem causing high costs, time and effort to map the data. Ontologies have set out to overcome not only the semantic issues but also lock-in problem by allowing the system to roll out semantically interoperable [17, 25]

These related works tend to focus on NoSQL databases that are distributed in the cloud environment. The latter is suitable for NoSQLs characteristics like large-scale data, availability, and scalability. However, the authors have not deducted the principal criteria that lead us to select the best alternative data model via the mapping from one NoSQL type to another.

As object relational databases play an important role in representing complex data, Aggoune and Namoune [23] recently proposed a new approach for data mapping from Oracle object-relational to MongoDB document-oriented database. The proposed approach aims to improve the processing of large amounts of complex data through a set of mapping rules applied between object relational and document-oriented schemata. This approach allows the system to maintain the integrity constraint of object-relational model against the schema-less of NoSQL databases.

This work represents a good solution to migrate the existing object-relational database to the new data format instead of creating data from scratch. Given the complexity of the object-relational database, it may be necessary to adapt this approach in the context of NoSQL data mapping.

5 A Study of the Recent Frameworks of NoSQL Data Mapping

The need to map between different NoSQL databases involves building frameworks, which are used extensively in practice. The available frameworks have been developed to ensure the mapping between NoSQL databases in the cloud environment [18].

CDPort (Cloud Data Portability) framework [24] provides a unified common API for ensuring the portability between different cloud-based NoSQL data. This framework supports Amazon SimpleDBs key-value data, MongoDBs document-oriented data, and Google Datastores columnar-oriented data. The NoSQL data mapping is based on three wrappers one for each NoSQL system to ensure the data transformation.

The SDCP (Service Delivery Cloud Platform) [40] represents a middleware infrastructure that uses resources from multiple cloud columnar-oriented databases for mapping between them.

In [26] a NoSQL data mapping framework has been proposed, which is based on a meta-model and a compression Algorithm.

Wijaya and Arman [45] have proposed a general framework, which combines three existing frameworks [26, 36, 39] to solve the data mapping problem with a different solution, characteristics, and properties. The developed framework includes mapping algorithms, mapping models, and mapping schemes of four NoSQL databases.

Table 4. Comparative study between NoSQL data mapping.

Criteria	Framework of Bansel et al. [26]	CDPort Framework [24]	Framework of Wijaya and Arman [45]	SDCP Framework [40]
NoSQL Databases	Document, Graph, Columnar	Key-value, column, Document	Four categories	Columnar databases
NoSQL DBMS	MongoDB, Azure Table, Neo4j	Google Datastore, Amazon SimpleDB, MongoDB	MongoDB, Redis, Neo4j, Hbase	SimpleDB and Azure Table
Dataset	Twitter	Generic	Twitter	Generic
Algorithm	MetaModel, Compression Algorithm	Adapter, CPort data model	ETL Transformation	Service-based Algorithm
PL	–	Java	–	Java
Mapping strategies	Direct and Intermediate	Intermediate	Direct	Direct
Interface	–	CDPorts API	Nodejs API	Java Persistence
Architecture	Repository	Record	Record	Service

We focus on these recent NoSQL mapping frameworks to establish a comparative study between them. This comparison is based on essential criteria (see Table 4).

NoSQL databases mapping frameworks are commonly provide a uniform interface and unified data model for various NoSQL databases. Each framework concerns some NoSQL stores and uses different API and algorithm. The general framework proposed in [45] is composed of existing frameworks in order to ensure the mapping between four NoSQL categories.

As a result, we will propose a new framework for mapping between these NoSQL stores without using any existing framework or tool. In addition, the approach that will be proposed aims to handle bidirectional conversion.

6 Conclusion

In this paper, we have presented a literature review for mapping between NoSQL databases with a comparison study of different recent frameworks. The utility to integrate a mapping tool in applications as a means to use the best alternative NoSQL database compared to the initial database.

According to this review, many parameters need to be taken into consideration, for example the domain of the data set, the volume of the dataset, the data type, and the degree of relationship between values. In comparison to related work, we would like to suggest a new method for NoSQL data mapping to ensure that the initial data have to be adapted to the format and structure of the target database.

References

1. Allegrograph. `https://allegrograph.com/products/allegrograph/`, accessed: 2020-08-02
2. Arangodb. `https://www.arangodb.com/`, accessed: 2020-08-15
3. Azure table. `https://azure.microsoft.com/en-us/services/storage/tables/`, accessed: 2020-08-02
4. Cassandra. `https://cassandra.apache.org/`, accessed: 2020-08-02
5. Couchdb. `https://couchdb.apache.org/`, accessed: 2020-08-02
6. Db-engine. `https://db-engines.com/`, accessed: 2020-08-15
7. Documentdb. `https://aws.amazon.com/fr/documentdb/`, accessed: 2020-08-02
8. Hbase. `https://hbase.apache.org/`, accessed: 2020-08-02
9. Hypergraphdb. `https://allegrograph.com/products/allegrograph/`, accessed: 2020-08-02
10. Ibm grap. `https://www.ibm.com/uk-en/marketplace/graph`, accessed: 2020-08-02
11. Marklogic. `https://www.marklogic.com/`, accessed: 2020-08-02
12. Memcached. `https://memcached.org/`, accessed: 2020-07-30
13. Mongodb. `https://mongodb.com/`, accessed: 2020-08-02
14. Oraclenosql. `https://www.oracle.com/database/technologies/related/nosql.html`, accessed: 2020-07-30

15. Redis. `https://redis.io/`, accessed: 2020-07-30
16. Riak. `https://riak.com/`, accessed: 2020-07-30
17. Aggoune, A.: Automatic ontology learning from heterogeneous relational databases: Application in alimentation risks field. In: IFIP International Conference on Computational Intelligence and Its Applications. pp. 199–210. Springer (2018)
18. Aggoune, A.: Switching between dierent nosql databases: Approaches and frameworks. IAM (2020)
19. Aggoune, A., Bouramoul, A., Kholladi, M.K.: Personnalisation daccès aux sources de données hétérogènes pour lorganisation des grands systèmes dinformation dentreprise. IT4OD p. 109 (2014)
20. Aggoune, A., Bouramoul, A., Kholladi, M.K.: Big data integration: A semantic mediation architecture using summary. In: 2016 2nd International Conference on Advanced Technologies for Signal and Image Processing (ATSIP). pp. 21–25. IEEE (2016)
21. Aggoune, A., Namoune, M.S.: From object-relational to nosql databases: A good alternative to deal with large data. CITCS (2019)
22. Aggoune, A., Namoune, M.S.: Practical study for handling of nosql data on the distributed environment systems. IAM (2019)
23. Aggoune, A., Namoune, M.S.: A method for transforming object-relational to document-oriented databases. In: 2020 2nd International Conference on Mathematics and Information Technology (ICMIT). pp. 154–158. IEEE (2020)
24. Alomari, E., Barnawi, A., Sakr, S.: Cdport: A framework of data portability in cloud platforms. In: Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services. pp. 126–133 (2014)
25. Andročec, D., Vrček, N.: Ontology-based resolution of cloud data lock-in problem. Computing and Informatics **37**(5), 1231–1257 (2018)
26. Bansel, A., González-Vélez, H., Chis, A.E.: Cloud-based nosql data migration. In: 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP). pp. 224–231. IEEE (2016)
27. Bathla, G., Rani, R., Aggarwal, H.: Comparative study of nosql databases for big data storage. International Journal of Engineering & Technology **7**(26), 83 (2018)
28. Cattell, R.: Scalable sql and nosql data stores. Acm Sigmod Record **39**(4), 12–27 (2011)
29. Corbellini, A., Mateos, C., Zunino, A., Godoy, D., Schiaffino, S.: Persisting big-data: The nosql landscape. Information Systems **63**, 1–23 (2017)
30. Davoudian, A., Chen, L., Liu, M.: A survey on nosql stores. ACM Computing Surveys (CSUR) **51**(2), 1–43 (2018)
31. Gessert, F., Wingerath, W., Friedrich, S., Ritter, N.: Nosql database systems: a survey and decision guidance. Computer Science-Research and Development **32**(3-4), 353–365 (2017)
32. Hajoui, O., Dehbi, R., Talea, M., Batouta, Z.I.: An advanced comparative study of the most promising nosql and newsql databases with a multi-criteria analysis method. Journal of Theoretical and Applied Information Technology **81**(3), 579 (2015)
33. Horii, H.H.: Query processing with bounded staleness for transactional mutations in nosql database (Jun 11 2019), uS Patent 10,318,521
34. Meier, A., Kaufmann, M.: Nosql databases. In: SQL & NoSQL Databases, pp. 201–218. Springer (2019)

35. Sahatqija, K., Ajdari, J., Zenuni, X., Raufi, B., Ismaili, F.: Comparison between relational and nosql databases. In: 2018 41st international convention on information and communication technology, electronics and microelectronics (MIPRO). pp. 0216–0221. IEEE (2018)
36. Scavuzzo, M., Di Nitto, E., Ceri, S.: Interoperable data migration between nosql columnar databases. In: 2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations. pp. 154–162. IEEE (2014)
37. Scavuzzo, M., Tamburri, D.A., Di Nitto, E.: Providing big data applications with fault-tolerant data migration across heterogeneous nosql databases. In: 2016 IEEE/ACM 2nd International Workshop on Big Data Software Engineering (BIGDSE). pp. 26–32. IEEE (2016)
38. Sharma, S., Tim, U.S., Gadia, S., Wong, J., Shandilya, R., Peddoju, S.K.: Classification and comparison of nosql big data models. International Journal of Big Data Intelligence **2**(3), 201–221 (2015)
39. Shirazi, M.N., Kuan, H.C., Dolatabadi, H.: Design patterns to enable data portability between clouds' databases. In: 2012 12th International Conference on Computational Science and Its Applications. pp. 117–120. IEEE (2012)
40. Silva, L.A.B., Costa, C., Oliveira, J.L.: A common api for delivering services over multi-vendor cloud resources. Journal of Systems and Software **86**(9), 2309–2317 (2013)
41. Tang, E., Fan, Y.: Performance comparison between five nosql databases. In: 2016 7th International Conference on Cloud Computing and Big Data (CCBD). pp. 105–109. IEEE (2016)
42. Thalheim, B., Wang, Q.: Data migration: A theoretical perspective. Data & Knowledge Engineering **87**, 260–278 (2013)
43. Van der Veen, J.S., Van der Waaij, B., Meijer, R.J.: Sensor data storage performance: Sql or nosql, physical or virtual. In: 2012 IEEE fifth international conference on cloud computing. pp. 431–438. IEEE (2012)
44. Vukotic, A., Watt, N., Abedrabbo, T., Fox, D., Partner, J.: Neo4j in action. Manning Publications Co. (2014)
45. Wijaya, Y.S., AkhmadArman, A.: A framework for data migration between different datastore of nosql database. In: 2018 International Conference on ICT for Smart Society (ICISS). pp. 1–6. IEEE (2018)
46. Xiang, P., Hou, R., Zhou, Z.: Cache and consistency in nosql. In: 2010 3rd International Conference on Computer Science and Information Technology. vol. 6, pp. 117–120. IEEE (2010)