Literature Review on Database Design Testing Techniques

Abdullahi Abubakar Imam[1,2](✉), Shuib Basri[1], Rohiza Ahmad[1], and María T. González-Aparicio[3]

[1] SQ2E Research Cluster, Computer and Information Sciences Department, Universiti Teknologi PETRONAS, Bandar Seri Iskandar, 32610 Seri Iskandar, Perak, Malaysia
aiabubakar3@gmail.com, {abdullahi_g03618, shuib_basri, rohiza_ahmad}@utp.edu.my

[2] Ahmadu Bello University, Zaria, Nigeria

[3] Computing Department, University of Oviedo, Gijon, Spain
maytega@uniovi.es

Abstract. Database driven software applications are becoming more sophisticated and complex. The behavior of these systems solely depends on the data being used. Whereas this data has now become so massive, varyingly connected, distributed, and stored and retrieved with different velocity, era of big data. To make these systems operate in every anticipated environment with the required usability, durability and security, they are subjected to rigorous testing using the available Software Testing Techniques (STT). This test is described as a process of confirming correct behavior of a piece of software which consists of three parts, namely, interface (GUI), back-end (codes) and data-source (database). The purpose of this study is to identify and analyze existing STT in the context of databases design structures. Primary studies related to ST were identified using search terms with relevant keywords. These were classified under journal and conference articles, book chapters, workshops, and symposiums. Out of the search results, 23 Primary studies were selected. Database testing has been significantly discussed in the software testing domain. However, it was discovered that, existing software testing techniques suffer from several limitations which includes: incompatibility with new generation databases, lack of scalability and embedded SQL query detection issues. In addition, application of existing techniques into a full-fledged software system has not been reported yet.

Keywords: Software testing · SQL databases · NoSQL databases · Software quality

1 Introduction

As software applications get more complex and intertwined and with increasing usage of massive and varieties of data from different sources, cybercrimes and varieties of platforms, it is more important than ever to have a durable, robust, secured and consistent database system [1]. Similarly, methodologies that are adopted to make sure that database driven applications being developed are absolutely tested and meet their specified requirements need to be vigorous and flexible such that software applications can be screened to successfully operate in every anticipated environment with the required usability and security [2]. If software is deployed and database operations such as Create, Retrieve, Update and Delete (CRUD) are performed without testing the application's database performance, security and reliability, the company risks a crash of the entire system, internal logical structure deteriorations or broken insertions, deletions or updates [3].

According to Sree [4], Software Testing (ST) can be defined as a process of identifying defects from a software, isolating them, and subjecting (sending) them for rectification. On the words of [5], software testing is a process of providing information about the quality of system to stakeholders by investigating the system under test. The process ensures that there is no single defect associated to the product through testing any of the three main software components, namely; interface, codes and database. For these reasons among others, techniques such as black box, white box and gray box testing came into view to test software applications through the user interface or internal logical structure or both respectively. Also, approaches to particularly test the SQL database driven applications were proposed since analyzing codes only is insufficient for threats detection and mitigation [6, 7].

Although these approaches have proven useful in detecting problems associated with software design and implementation and SQL database operations as well as security problems such as SQL injection and cross-origin attacks, they should be adjusted to detect specific vulnerabilities database driven software applications [7].

In this article we analyzed and compartmentalized the existing studies with respect to software testing techniques in the context of database design structure. Primary studies related to ST were identified using search terms with relevant keywords. These were classified under journal and conference articles, book chapters, workshops, and symposiums. Out of the search results, 23 Primary studies were selected. Database testing has been significantly discussed in the software testing domain. However, it was discovered that, existing software testing techniques suffer from several limitations which includes: incompatibility with new generation databases, lack of scalability and embedded SQL query detection issues. In addition, application of existing techniques into a full-fledged software system has not been reported yet.

The remainder of this paper is structured as follows: Sect. 2 explains the method adopted. Section 3 presents the results. Section 4 discussed the findings. Section 5 highlights some recommendations. Section 6 concludes the paper.

2 Research Method

In this section, the method adopted to conduct this research is presented. It consists of research questions, literature sources and study selection process.

2.1 Research Questions

To commence the investigation about the state of the art in this area, three research questions were devised and put into words as follows:

RQ1 What are the existing software testing techniques, models or algorithms used to assess the storage (database) component of software?
RQ2 How do the identified solutions detect the menace associated with database design and what are their strengths and weaknesses?
RQ3 How to mitigate an existing technique to comprise the aspect of big data datasource as part of the testing components of a software application?

2.2 Literature Sources

In this study, a comprehensive and detailed search was conducted using the electronic libraries available such as Science Direct, Web of Science, IEEE Xplore, Springer, Google Scholar and ACM. These libraries were used to search for the relevant materials across the globe. The search yield several categories of materials starting from symposiums, conference proceedings, book chapters as well as journals papers.

2.3 Study Selection

Study selection is one of the key components of SLR and it's done during or after search process is completed, as such, a set of rules are engineered and applied to appropriately select the right studies. The rules are, manuscript can only be selected if it's:

- tackling any of the key words of this research.
- tackling any of the questions in this research or attempt to describe its nature.
- either published or submitted to a journal or conference. Book chapters as well as technical reports are also considered.
- written in English or fully translated to English language.
- related to topics such as SQL and NoSQL database evaluation and testing, software engineering and applications, software testing, CRUD operations testing, open source software development, teaching and education.

3 Results

The results of this study are presented in this section. These results are categorized into two different categories. We started by presenting the results related to software testing as a whole. In the second part, the results for SQL and NoSQL database testing are presented.

3.1 Software Testing Techniques

The following table (Table 1) presented the available software testing techniques in a summery form.

Table 1. Testing techniques

Components testing	Unit testing	Verification (Process Oriented)	**White box Testing** (Tests that are derived from knowledge of the program's structure and implementation)
	Module testing		
Integrated testing	Sub-system testing		
	System testing		
User testing	Acceptance testing	Validation (Product Oriented)	**Black Box Testing** (Tests are derived from the program specification)
Components testing + Integrated testing + User testing	Unit, module, system, sub-system and acceptance testing	Verification and validation	**Gray Box Testing** (Test are derived from both the knowledge of the program structure and specification)

In black-box testing, errors such as halting and a testing dependent functions independently, the functions may fail when the original sequence is changed at run time during the execution of other functions. As such, it is indispensable to consider a technique beyond black box testing.

However, white-box does not explicitly consider SQL statements which are embedded in application programs. It treats SQL statements as black boxes. Commonly, SQL semantics are not intentionally included in the test cases. Therefore, it is believed that, faults related to the internal database changes might be missed by traditional white box testing [8].

On the other hand, because gray-box testing technique is based on black-box and white-box testing techniques, it inherits all the problems associated with its parents (black-box and white-box) without exclusions. Although it combined several functionalities in one place, it is measured tedious, cumbersome and time consuming.

In view of the above, it can be concluded that the authors focus mainly on testing software applications through the user interface by adopting black box technique while others gave much emphasis to the coding side where white box technique is most suited. Alternatively, some authors believe that when the two techniques are combined, more reliable testing can be achieved. However, all these techniques and their associated works are only concentrating on the software itself while neglecting its datasource which contributes significantly to the quality and performance of any database driven software application. The following section presents the existing approaches for testing design structure of SQL and NoSQL databases driven applications.

3.2 SQL and NoSQL Database Testing

Table 2 below presents the approaches used to test the database driven software applications with respect to SQL and NoSQL databases.

In consideration of the related works on SQL database testing, it can be concluded that relational/traditional database have received considerable attention where several techniques with various approaches are reported. However, these techniques are either

Table 2. SQL & NoSQL database testing approaches

Author & Year	Techniques/Solution	Strength	Weaknesses
Tsumura et al. (2016)	Plain Pairwise Coverage Testing (PPCT) and Selected Pairwise Coverage Testing (SPCT)	No predicates Uses few elements is SQL query	–Limited test cases Only RDBMS
Hamlin and Herzog (2014)	SPAR Test Suite Generator (STSG)	Test suits are generated automatically Supports benchmarking and correctness-checking	–Relies on user specifications –Only RDBMS –No model checking
Setiadi and Lau (2014)	Data consistency framework based on system specifications	Data inconsistency detection	–Only RDBMS –Through GUI only –Starts after the execution of applications
Zou (2014)	Cloud storage area	Distribute data among the available computing devices	–Only RDBMS –Data privacy & security
Sarkar, Basu, and Wong (2014)	Novel framework called iConSMutate	Reusing existing DB states Generate test cases automatically	–Only RDBMS –No constraints solving –No model checking
Marin (2014)	Data-agnostic framework for test automation	Coverage model and fault model for OLAP cubes Uses record-and-replay	–OLAP –RDBMS
Setiadi and Lau (2014a)	Structured model that enables the automatic generation of consistency rules	Derive the data consistency rules from business rules, system specifications and database schema	–Only RDBMS –Can’t apply on existing system –Single repository
Grechanik, Hossain, and Buy (2013)	Novel approach for Systematic Testing in Presence of Database Deadlocks (STEPDAD)	Based database deadlocks	–Only RDBMS –No data privacy –No constraints solving
Pan, Wu, and Xie (2013)	The use of mutation scores to test the fault-detection capabilities	Ability to detect faults in real world database applications	–Only RDBMS
McCormick II et al. (2012)	MutGen to generate test for mutation testing for database applications	Ability to kill the mutant Detect DB constraints	–Only RDBMS

(*continued*)

Table 2. (*continued*)

Author & Year	Techniques/Solution	Strength	Weaknesses
Ron, Shulman-peleg, and Ibm (2016)	Comparative study of NoSQL DBs based NoSQL injections	Proved NoSQL security vulnerabilities	Awareness of NoSQL injections only. No measures
Gonzalez-Aparicio et al. (2016)	Context-aware model for CRUD operations	Trade-off analysis between availability & consistency of NoSQL DBs	Monitors the execution of CRUD operations for best selection of NoSQL DBs only
Bhogal and Choksi (2015)	Comparison study of NoSQL databases	NoSQL DBs evaluated based on variety of data	Volume, Velocity and veracity not considered
Truica et al. (2015)	Performance evaluation on CRUD operation for NoSQL databases	Ability to analyze features of document oriented databases	Compared 3 document oriented DBs (mongoDB, couchDB, & Couchbase) only
Klein et al. (2015)	Method to perform technology selection among NoSQL DBs	NoSQL databases are evaluated in a specific context	Not testing any BD variable Used to select NoSQL DB only
Abramova, Bernardino, and Furtado (2014)	Analysis of scalability properties of cloud serving NoSQL engine	Ability to test put, get and scan operations	Focus on scalability only
Naheman (2013)	Comparison between NoSQL and SQL DBs, and between NoSQL products	Performance testing on HBase	Comparison between SQL & NoSQL DBs only

embracing white-box technique (i.e. codes) or providing solutions that can work with SQL based database designs only. Also, the solutions are independent of software applications, so, operations like CRUD, security, database connection and data access schemas incorporated with application codes are not verified, thus incomplete testing of software.

Oppositely, it can be seen (from Table 3) that NoSQL databases are described as poor in the areas such as consistency, reliability, technical support and hardware/software compatibility which proves the use of BASE rather than ACID. Because of its complexity, insecurity, distributed nature of data stores and very high storage capacity, there is need to confirm its design, codes and implementation properly before system deployment [22, 23]. Though, techniques or approaches that focus on testing the big data (NoSQL) database driven software applications are yet to be reported.

Therefore it remains challenging to produce an algorithm, method, technique or model that can be used to test software applications with big data category of data-source, in particular, to be able to test big data (NoSQL) databases driven software applications with respect to NoSQL dependent variables such as CRUD loading time, security, data models, privacy, scalability and variability among others [23]. This is highly imperative as the databases are becoming larger and complex [20].

4 Results and Discussion

According to [24, 25], database testing refers to the analysis of database system's performance, consistency, availability, reliability and security. This test is usually independent of the application and is done to verify the ACID or BASE properties of a database management system. It consists of process that are layered in nature such as business layer, user interface layer, database itself and most importantly data access layer which deals directly with the database. Data access layer is the layer that is used for testing strategies like quality assurance and quality control. There are four stages to test database, namely: (1) Set fix, (2) Test run, (3) Outcome Verification and, (4) Tear down [24]. The solutions proposed for database testing are categorized based on SQL and NoSQL databases some of which are discussed below.

4.1 SQL Database Testing Techniques

This Define Relational databases are based on a set of principles to optimize performance which are derived from Consistency, Availability, Partition tolerance (CAP) theorem [26]. ACID which stands for Atomicity, Consistency, Isolation, and Durability, is a principle based on CAP theorem and used as set of rules for relational database transactions [26].

In line with Subsection 2.1.2 of this section where white-box testing is discussed, [8] believed that the same technique can be extended to cover database aspect of software applications. As such, WHODATE approach for database testing is proposed to semantically reflect SQL statements in the test case. As shown in the following figure, SQL queries are transformed into the traditional white box testing for test case generation. This is only applicable to SQL based databases as shown in Fig. 1.

At first, SQL queries are retrieved and passed to transformation unit where the queries are translated to best suit the format of white box technique. Thereafter, the white box generates its test cases as usual.

While [16] proposed the use of mutation scores to test the fault-detection capabilities of the real world database applications such as MySQL, SQL server and Oracle. Using this approach, sample schemas are selected from databases, and then queries from the schemas are run against the mutation operators. On the contrary, MutGen was introduced to generate test for mutation testing on database applications [15]. It incorporates weak-mutant-killing constraints in the code which queries mutant-killing constraints into the transformed code. This is done to kill the mutants by generating program inputs and sufficient database states. In contrast, [12] believed that the higher the mutation score the better because it indicates higher quality in identifying programming errors, thereby produced iConSMutate to generate test cases which will reuse the existing database state for database applications to improve quality in codes coverage and mutant detection. It should be noted that the mutant concept is only applicable to relational databases rather than the NoSQL databases.

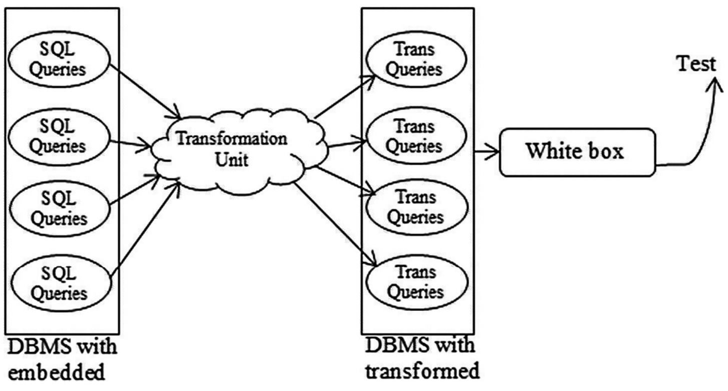

Fig. 1. WHODATE approach for SQL statement transformation

It is also discovered that After-State Database Testing (ASDT) can be used to detect data inconsistency in traditional databases only [10]. It is a black box based framework that uses design choices and system specifications to test the consistency state after the execution of the application. Using ASDT, a structured model that enables the generation of data consistency rules from business rules, database schema and system specifications. Its main aim is to minimize the use of resources and efforts in ASDT.

On the word of [6], multi-dimensional databases (OLAP cubes) can be tested when compared with record-and-replay technique using show-case. The solution is called data-agnostic which works together with coverage model and fault model for OLAP cubes. This is only applicable to multi-dimensional databases.

Contrariwise, since test suits are often produced and used for testing purposes, [9] produced SPAR Test Suite Generator (STSG) to automate the process of generating test suits used by SQL style database systems. Ground-truth answers are incorporated in the produced test suite. This approach supports benchmarking and correctness-checking on SQL based databases.

In [1] believed that avoiding the use of predicates while testing databases can yield more promising results. In substitute, two approaches were proposed: Plain Pairwise Coverage Testing (PPCT) and Selected Pairwise Coverage Testing (SPCT). These approaches use parameters selected from elements in the SQL SELECT and they are only applicable to SQL queries.

In consideration of the related works on database testing, it can be concluded that relational/traditional database have received considerable attention where several techniques with various approaches are reported. However, these techniques are either embracing white-box technique (i.e. codes) or providing solutions that can work with SQL based database designs only. Also, the solutions are independent of software applications, so, operations like CRUD, security, database connection and data access schemas incorporated with application codes are not verified, thus incomplete testing of software.

4.2 NoSQL Database Testing Techniques

NoSQL (not only SQL) is a trending term in modern data stores that accommodate big data; it refers to non-relational databases that rely on different storage mechanisms such as document store, key-value store, columner and graph [7].

Unlike the SQL databases that based on ACID, Not Only SQL (NoSQL) focuses on BASE principle in order to have robust and correct database with huge amount of data [7, 26]. BASE stand for Basically Available, Soft state, Eventually consistent. BASE follows CAP theorem and two of three guarantees must be chosen if the system is distributed (Brewer [27]).

[7] discovered that, the aforesaid NoSQL databases are highly vulnerable to cross-origin attacks. Since NoSQL databases use different query languages which make SQL injections irrelevant, attackers find a new opportunity to insert malicious code easily. Therefore, it's recommended to prevent all possible attacks via careful code and database examination before complete system deployment. Although the existing tools and techniques have proven useful in detecting SQL injection attacks, they should be adjusted to detect the specific NoSQL database vulnerabilities [7].

However, it is believed that, CRUD operations can be used to test data availability and consistency of each of the NoSQL databases. Hence [17] proposed a context-aware model for CRUD operations to monitors the execution of CRUD operations for best selection of NoSQL databases. Although this will give programmers the chance to make appropriate choice that corresponds to the client needs, it does not have the capability of detecting any erroneous practice in the logical structure of the application code that communicates with the designed database. Errors such as functions call order cannot be detected.

In the work of [18], a comparison was conducted between the NoSQL databases based on data variety as one of the 5, 6 or 7 Vs of big data. While in [19] the comparison was made between the document oriented NoSQL databases (MongoDB, CouchDB, and Couchbase) for asynchronous replications on CRUD operations. The outcome of the test was compared with famous relational databases such as Microsoft SQL Server, MySQL, and PostgreSQL to measure the scalability and flexibility of systems. Nevertheless, the two inventions were only making comparison between the existing SQL and NoSQL databases not the design principles of particular database driven software application.

In contrast, [20] considered all the NoSQL data models (key-value, column, document, graph) in testing a particular healthcare system requirements. One database was considered from each of the NoSQL data model category (Riak, Cassandra and MongoDB) with exception of graph as it does not support horizontal partitioning. A systematic method to perform this technology selection is proposed to evaluate the products in the specific context of use which do not focus on database design principles violations.

It is reported that existing NoSQL databases are not matured enough [22], which is why it is necessary to examine all the NoSQL database features based on system requirements before selecting one [23]. In [22], performance testing was conducted on the NoSQL databases, HBase in particular, for relationship mapping between number of column families and performance.

Table 3. Comparison between NoSQL and SQL Databases Naheman [22]

	NoSQL databases	SQL databases
Read & write concurrency	Fast	Slow
Data storage	Mass storage	General storage
Expansion mode	Horizontal expansion	Vertical expansion
Consistency	Poor	Good
Reliability	Poor	Good
Availability	Good	Good
Expansion costs	Low	High
Maturity	Low	High
Programmatically	Complex	Simple
Data mode	Free	Fixed
Human overhead	Relatively high	Medium
Technical support	Poor	Good
Upgrade costs	High	Low
Hard/software compatibility	Poor	Good
Flexibility	Good	Poor

It can be seen that NoSQL databases are described as poor in the areas such as consistency, reliability, technical support and hardware/software compatibility which proves the use of BASE rather than ACID. Because of its complexity, insecurity, distributed nature of data stores and very high storage capacity, there is need to confirm its design and implementation properly before system deployment [22, 23].

5 Findings and Recommendations

In general, this research has produced several findings as presented in Sects. 3 and 4. However, for quick reference, some findings are summarized as follows:

- Database Logical Structure: A technique that will detect incorrect logical implementation of CRUD operations and security measures based on the predefined functional and non-functional requirements is needed.
- Auto-problem solving technique: A technique that will suggest proper approach of restructuring CRUD operations and security measures based on the best practice.
- A technique that will propose structural amendments for better system performance and better data security approach.

- No indexing of files: the use of Btree selection concept to group and sample appropriate files out of the pool of distributed file will not only make the testing accurate but it will also make the testing process faster.
- Slow processing: there is need to establish a new approach for sorting files from smallest to the largest to enhance records searching capabilities during test.
- Security: existing techniques need to be revisited for security problems such as SQL injection and cross-origin attacks [7].

Erroneous database design: As data becomes huge, the loading time gets affected proportionately [6]. So, there is need to test software applications with varieties of data sizes to confirm the correct behavior of a system.

6 Conclusion and Future Focus

In this article we analyzed and compartmentalized the existing studies with respect to software testing techniques in the context of database design structure. Primary studies related to software testing were identified using search terms with relevant keywords. These were classified under journal and conference articles, book chapters, workshops, and symposiums. Out of the search results, 23 Primary studies were selected. Database testing has been significantly discussed in the software testing domain. However, it was discovered that, existing software testing techniques suffer from several limitations which includes: incompatibility with new generation databases, lack of scalability and embedded SQL query detection issues. In addition, application of existing techniques into a full-fledged software system has not been reported yet.

Acknowledgment. This paper/research was fully supported by Ministry of Higher Education Malaysia, under the Fundamental Research Grant Scheme (FRGS) with Ref. No. FRGS/1/2018/ICT04/UTP/02/04.

References

1. Tsumura, K., Washizaki, H., Fukazawa, Y., Oshima, K., Mibe, R.: Pairwise coverage-based testing with selected elements in a query for database applications. In: 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops, pp. 92–101 (2016)
2. Inflectra, B.: Software Testing Methodologies (2016). https://www.inflectra.com/Ideas/Topic/Testing-Methodologies.aspx. Accessed 01 Jan 2016
3. Reza, H., Zarns, K.: Testing relational database using SQLLint. In: Proceedings - 2011 8th International Conference on Information Technology: New Generations, ITNG 2011, pp. 608–613 (2010)
4. Sree, U.: Software Testing Life Cycle: Defects and Bugs (2016). https://olaiainforarch.wordpress.com/. Accessed 11 July 2016
5. Berger, D., Fröhlich, P.: Software testing techniques. Power Point Lecture, 20 pages (2016)
6. Marin, M.: A data-agnostic approach to automatic testing of multi-dimensional databases. In: Proceedings of - IEEE 7th International Conference on Software Testing, Verification and Validation, ICST 2014, pp. 133–142 (2014)

7. Ron, A., Shulman-Peleg, A., Puzanov, A.: Analysis and mitigation of NoSQL injections. IEEE Secur. Priv. **14**(2), 30–39 (2016)
8. Chan, M.Y., Cheung, S.C.: Testing database applications with SQL semantics. In: Proceedings of 2nd International Symposium on Cooperative Database Systems for Advanced Applications, March, pp. 363–374 (1999)
9. Hamlin, A., Herzog, J.: A test-suite generator for database systems (2014)
10. Setiadi, R., Lau, M.F.: Identifying data inconsistencies using after-state database testing (ASDT) framework. In: Proceedings of the International Conference on Quality Software, pp. 105–110 (2014)
11. Zou, J.: Research and application of testing technology of the cloud computing database. In: Proceedings - 2014 IEEE Workshop on Electronics, Computer and Applications, IWECA 2014, pp. 699–702 (2014)
12. Sarkar, T., Basu, S., Wong, J.: IConSMutate: concolic testing of database applications using existing database states guided by SQL mutants. In: Proceedings of 11th International Conference on Information Technology: New Generations, ITNG 2014, pp. 479–484 (2014)
13. Setiadi, R., Lau, M.F.: A structured model of consistency rules in After-State Database Testing. In: 38th IEEE International Computer Software and Applications Conference Workshops, no. 2, pp. 650–655 (2014)
14. Grechanik, M., Hossain, B.M.M., Buy, U.: Testing database-centric applications for causes of database deadlocks. In: Proceedings - IEEE 6th International Conference on Software Testing, Verification and Validation, ICST 2013, vol. 191242, pp. 174–183 (2013)
15. Pan, K., Wu, X., Xie, T.: Automatic test generation for mutation testing on database applications. In: 8th International Workshop on Automation of Software Test (AST), pp. 111–117 (2013)
16. McCormick II, D.W., Frakes, W.B., Anguswamy, R., McCormick, D.W.: A comparison of database fault detection capabilities using mutation testing. In: 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 323–326 (2012)
17. Gonzalez-Aparicio, M.T., Younas, M., Tuya, J., Casado, R.: A new model for testing CRUD operations in a NoSQL database. In: 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), vol. 6, pp. 79–86 (2016)
18. Bhogal, J., Choksi, I.: Handling big data using NoSQL. In: Proceedings - IEEE 29th International Conference on Advanced Information Networking and Applications Workshops, WAINA 2015, pp. 393–398 (2015)
19. Truica, C.O., Radulescu, F., Boicea, A., Bucur, I.: Performance evaluation for CRUD operations in asynchronously replicated document oriented database. In: Proceedings - 2015 20th International Conference on Control Systems and Computer Science, CSCS 2015, pp. 191–196 (2015)
20. Klein, J., Gorton, I., Ernst, N., Donohoe, P., Pham, K., Matser, C.: Performance evaluation of NoSQL databases: a case study. In: Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems, pp. 5–10 (2015)
21. Abramova, V., Bernardino, J., Furtado, P.: Testing cloud benchmark scalability with cassandra. In: 2014 IEEE World Congress on Services, pp. 434–441 (2014)
22. Naheman, W.: Review of NoSQL databases and performance testing on HBase. In: 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer, pp. 2304–2309 (2013)

23. Cai, L., Huang, S., Chen, L., Zheng, Y.: Performance analysis and testing of HBase based on its architecture. In: IEEE 12th International Conference on Computer and Information Science (ICIS), 2013 IEEE/ACIS, pp. 353–358 (2013)
24. Henry, K.: Database System Concepts. Macgraw-Hill, New York (2010)
25. Silberschatz, S., Korth, Sudarshan: Database System Concept: Homogeneous Distributed Databases. Cent. Wiskd. Inform., pp. 19.3–19.125 (2007)
26. Abramova, V., Bernardino, J.: NoSQL databases: MongoDB vs cassandra. In: Proceedings of the International C* Conference on Computer Science and Software Engineering ACM 2013, pp. 14–22 (2013)
27. Brewer, E.: CAP twelve years later: how the 'rules' have changed. Comput. (Long. Beach. Calif) **45**(2), 23–29 (2012)