Automated Assessment of Student Queries in Redis

Yuliia Prokop
Department of Computer Science, Czech Technical University in Prague, Jugoslávských partyzánů 1580/3, 160 00 Prague, Czech Republic
prokoyul@fel.cvut.cz

Keywords: NoSQL, Redis, Key-value Store, Automated Assessment, RedisJSON, RediSearch, Education, Databases.

Abstract: In recent years, the popularity of NoSQL systems has grown significantly due to their flexibility and high performance when working with large volumes of data. Redis is one of the most popular key-value stores actively used in industry and education. However, automated approaches for NoSQL assignment evaluation, especially those involving advanced Redis modules, remain underdeveloped. This paper presents a web-based system for automated assessment of students' Redis queries, supporting basic structures (e.g., list, sorted set, and hash manipulation) and advanced features (RedisJSON and RediSearch). The system provides instant feedback on syntax errors, enabling students to correct mistakes and resubmit solutions in real-time. A pilot study with 42 master's students showed that about 78% successfully mastered the basics of Redis on the first try, while only 39% passed advanced assignments. With repeated attempts and targeted feedback, overall success on advanced tasks increased to 76%, highlighting the importance of continuous, automated guidance. The paper also discusses typical errors logged by the system (inconsistent or incorrect key naming, syntax errors when setting key expiration, incorrect index creation or JSON field references, etc.). The results demonstrate that integrating an automated Redis query assessment into educational programs can significantly enhance student engagement and learning efficiency. The flexible and modular design of the proposed system allows easy extension to other NoSQL databases and provides valuable data for instructors to refine teaching materials.

1 INTRODUCTION

Digital transformation drives the continuous development of data processing technologies and generates new technical challenges. Over the last decade, the increased demand for efficient handling of big, unstructured, and semi-structured data has led to the active development of NoSQL database management systems. For instance, the rapid growth of real-time analytics and IoT applications underscores the need for more flexible and scalable data solutions.

According to DB-Engines Ranking [1], as of February 2025, relational DBMSs still occupy about 72% of the market, while the remaining 28% are represented mainly by NoSQL solutions. At the same time, the demand for specialists familiar with non-relational DBMSs is growing, which calls for modernizing the curricula in technical universities. In response, many educational programs now integrate specialized modules on NoSQL technologies to better equip students for emerging industry demands. The key direction of modernization is the development of automated knowledge assessment systems that allow students to get real-time feedback when performing database tasks.

Despite developing tools for validating SQL and some document-oriented databases, key-value stores remain underrepresented in automated evaluation. Redis, one of the most popular in-memory solutions, is rarely considered a full-fledged teaching and validation object, especially for its advanced features (RedisJSON, RediSearch). This paper fills this gap by proposing a methodological and software toolkit for the automated assessment of Redis skills.

2 LITERATURE REVIEW

Many modern studies indicate a steady trend toward integrating NoSQL technologies into academic database programs [2 – 9]. The practice-oriented (hands-on) approach is the most widely used. It helps

students to deepen their understanding of modern data storage and processing methods.

The modern ecosystem of non-relational DBMSs includes several major models (key-value stores, document stores, wide-column stores, and graph databases) [2, 10]. Redis is ranked 6th among all databases in the DB-Engines ranking and is widely used in the industry. It is often demonstrated to students as an in-memory NoSQL system showing the basic concepts: caching, message brokering, and real-time data processing. Some studies [2 – 4] confirm that integrating Redis into academic courses helps students better understand NoSQL's theoretical aspects and acquire skills relevant to modern industrial tasks.

Some authors study the problems of evaluating student queries in NoSQL. Thus, the paper [11] describes the peer correction technique: students assess their colleagues' queries, identifying syntactic and semantic errors in SQL and NoSQL. The authors show that the average accuracy of such evaluations reaches 83%, and the correction results correlate moderately with the overall performance. The authors conclude that peer correction contributes to a better understanding of one's own errors and the development of query writing skills.

The paper [12] presents a TriQL system designed to teach three types of databases at once: relational (MySQL), graph (Neo4j), and document-oriented (MongoDB). Student work is assessed through lab assignments in which students compare the results of their queries executed on native engines. Such an approach helps to analyze the differences in depth and realize the advantages and limitations of each model. The authors of the paper [13] explore the possibilities of automatically evaluating student queries in SQL, MongoDB, and Neo4j using a combination of the PrairieLearn platform and proprietary analysis algorithms.

The articles [14, 15] describe a similar mechanism. Students' solutions in MongoDB and graph databases are executed through the PrairieLearn online platform, which automatically matches the results to a reference solution. Works [16, 17] describe the integration of the NoSQL Data Adapter module into the Moodle system, allowing students to perform queries remotely on NoSQL databases and receive instant automatic feedback.

Analysis of current research shows that many authors focus on pedagogical aspects of learning and do not study query assessment automation. In addition, these works rarely address key-value systems.

Existing platforms for automatic evaluation of student queries predominantly lack built-in NoSQL support. Some specialized solutions exist, e.g., MongoDB University (https://learn.mongodb.com/), Neo4j Sandbox (https://neo4j.com/sandbox/), DataStax Academy for Cassandra (https://www.datastax.com/dev/academy), and Redis University (university.redis.com). They provide interactive tools to teach mostly only basic operations, and the possibilities to integrate them into the classroom are limited.

Thus, integrating NoSQL into university curricula and searching for efficient automatic evaluation of students' assignments are topical problems that attract the attention of modern researchers. However, key-value stores – particularly Redis – do not receive sufficient attention in most works. The methodological aspects of teaching and automatic assessment of Redis's advanced features (hash and JSON data indexing, full-text search, aggregation, etc.) have also been insufficiently studied and require further research.

This paper aims to develop and justify a specialized methodological and software toolkit for the automated assessment of student queries in Redis with instant feedback and the ability to correct errors in the same session.

3 METHODS AND MATERIALS

3.1 General Solution Architecture

As part of the research, we developed a web application to automate assessing students' knowledge of using Redis. Its main components are:

- Web interface (client part) for entering queries and viewing results;
- Server part, which processes incoming queries and interacts with Redis;
- Redis NoSQL server, where commands are executed (taking into account RedisJSON and RediSearch modules);
- A database (MongoDB) for logging and storing the validation results.

In designing the application, we emphasized a modular architecture that facilitates straightforward updates and expansions. Instructors can modify or add functionalities by ensuring each component is loosely coupled without impacting other system parts.

The architectural diagram of the system is presented in Fig. 1.

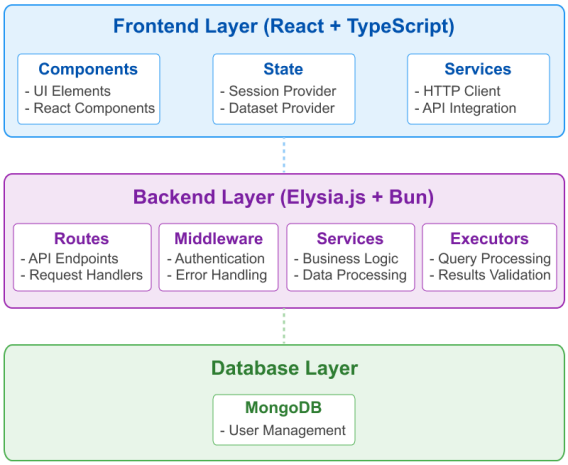


Figure 1: Architectural diagram of the system (source: author's development).

The interaction of components takes place according to the scheme "client-server-Redis-MongoDB." The web interface accesses the server part, which forms and forwards requests to Redis. After that, the results are returned for display to the student, and at the same time, logs are written to MongoDB.

3.2 Materials and Data Preparation

A dataset covering all key Redis types – strings, lists, sets, sorted sets, hashes, bit fields, and geodata – has been developed to ensure comprehensive testing. It includes about 200 queries for inserting various information about stores, products, and customers, such as:

```
HSET product:1 name "Espresso" price
2.50 calories 3 caffeine 64 size "1 oz"
ZADD store:1:menu 1 "Espresso" 2
"Latte" 3 "Cappuccino" 4 "Americano" 5
"Mocha" 6 "Flat White" 7 "Cold Brew" 8
"Frappe" 9 "Green Tea" 10 "Chai Latte"
LPUSH store:1:recent_orders "Vanilla
Latte" "Espresso" "Caramel Macchiato"
"Americano" "Mocha" "Cappuccino" "Flat
White" "Cold Brew" "Frappe" "Green Tea"
SADD loyalty:Gold customer:1 customer:4
GEOADD store_locations -122.2585
37.8614 "CityBrew University"
```

Each task is stored as a JSON document and includes:

- task description (context and required actions);
- a reference solution (one or more correct queries);
- test query (optional – provided only when the operation does not return a result).

Queries are grouped in collections according to the topics to which they relate.

Example of a task JSON document (from the data selection collection):

```
{
"Question": "Get the store with the
lowest sales for the current month.",
"Solution": ["ZRANGE monthly_sales 0 0
WITHSCORES"],
"Test": ""
}
```

Example of a task from the RediSearch collection:

```
{
"Question": "Create an index and find
users with Gold loyalty status. Return:
name, email, loyalty_status. Sort by
'name' ascending. Use LIMIT 0 1000.",
"Solution": [
"FT.CREATE
idx:customer_loyalty_only_gold ON HASH
PREFIX 1 customer: SCHEMA
loyalty_status TEXT SORTABLE name TEXT
SORTABLE email TEXT SORTABLE",
"FT.SEARCH
idx:customer_loyalty_only_gold
'@loyalty_status:Gold' SORTBY name ASC
LIMIT 0 1000 RETURN 3 name email
loyalty_status"],
"Test": ""
}
```

The teacher creates tasks by combining different data structures and scenarios. The level of task complexity in the system is not directly fixed. If necessary, the teacher can flexibly control the "depth" and complexity of queries, forming a variety of task collections.

For more precise control over the progress of learning, the tasks are divided into two main groups: basic operations (creation, modification, selection, deletion, simple aggregates) and advanced functions (RedisJSON, RediSearch): index creation, full-text search, aggregation, and work with complex JSON structures.

3.3 Methods of Assessment and System Operation

The algorithm for giving and executing assignments consists of these steps:

1) Selecting an assignment group: The student indicates whether they want to do basic or advanced assignments in the web interface.
2) Task list generation: Based on the teacher's settings, the system randomly generates a set of assignments with the required number of tasks.
3) Task display: On the web application page, the student sees the text of the current task and an area to enter a Redis query (or queries).
4) Entering and executing: After entering the query text, the student clicks the "Execute" button; they can skip the question by clicking the "Skip" button.

The system performs the following steps automatically:

1) Parsing and splitting queries: The text the user sends as a single answer to the current question is split into separate Redis commands.
2) Resetting the test environment: Before each test query, the initial dataset in Redis is recreated anew, and all previously created indexes are deleted. This guarantees the same initial database state.
3) Student query execution: The system sends commands to the Redis server. If Redis returns a syntax error, the user is notified immediately, and the testing process is paused. The student can correct and send the query again for execution (return to step 1).
4) Comparison with the reference solution: If the query is executed without errors, the system resets the data to its original state and executes the reference query. The results (full text) are then compared with the student query.
5) Saving logs: The testing process (task description, reference solution and result, student's solution and result, and comparison result in the form of true or false) is recorded in MongoDB.
6) Final result output: The system provides the student with a message listing the number of correct queries, a list of incorrect queries, and a notification about the test's success.

The result of each task is considered correct if the final state of the data in Redis or the returned result matches the state of the data or the result of the reference solution.

The system uses a full-text comparison of the output results of two solutions (user and reference). Each task description is designed to avoid evaluation ambiguities so that the final database state or query result is clearly defined once all requirements are met. For example, the assignment explicitly specifies the sorting order of the output records, the list of fields to include in the result, the numeric data types for the fields in the created index, and the record limit. This ensures high evaluation accuracy.

However, despite its simplicity and reliability, this approach has certain limitations. To address these, it is planned to introduce structure-oriented comparison and more flexible matching rules (e.g., ignoring character case and the order of records in the result or supporting incomplete matches). These improvements will extend the system's functionality and allow a more accurate evaluation of students' queries, not limiting them to a rigid full-text check.

The block diagram of the algorithm for automatic query evaluation is shown in Fig. 2.

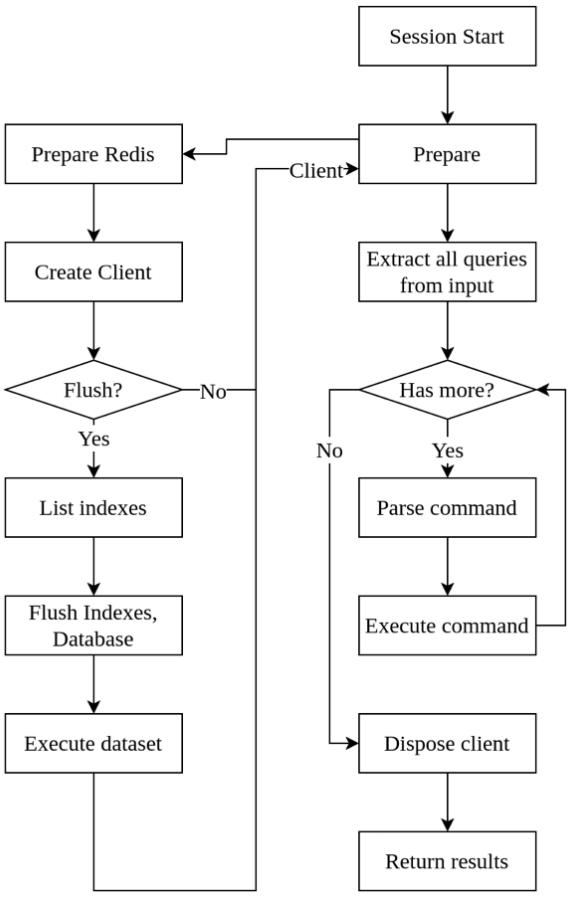


Figure 2: Block diagram of the algorithm for automatic query assessment (source: author's development).

The teacher sets the minimum number of correct solutions required to pass the test. Based on these settings, a notification about the test's success is generated. If the student falls short, they can retake the test.

3.4 User Interface

The web application implements authentication via login and password matching the credentials of the NoSQL server. From a cybersecurity perspective, this approach restricts access solely to enrolled university students, as external registration is disabled. By pre-registering accounts on the NoSQL server – managed by the instructor before the semester begins – the system minimizes the risk of unauthorized access and potential security breaches, ensuring that only verified and trusted users can interact with the testing environment.

After logging in, the following functionality is available to the students:

- selection of test type ("Redis basic" or "Redis advanced" buttons);
- list of current tasks (issued sequentially);
- field for entering a query;
- "Execute" button;
- "Skip" button (in case the input field is empty);
- "Open dataset" button;
- "Quit" button to abort the test;
- displaying Redis syntax errors (if any).

Figure 3 shows a screenshot of the window while testing Redis advanced queries. The "Execute" button is inactive until the answer is entered. The gray text in the answer field reminds users that queries should be separated by a blank line. The system also allows the assessment of queries in MongoDB, and this functionality is currently being tested.

Figure 3: Screenshot of the user window when taking the test (source: author's development).

After the test is completed, the student receives final feedback. Skipped tasks are marked as incorrect and added to the list of incorrect queries.

The web interface of the teacher (Fig. 4) provides the following features:

- select a specific test kind and load a collection of tasks;
- configure the number of tasks from each collection that will be included in the test;
- delete a collection of tasks.

The administrator loads the data set (a set of commands for inserting data of different types). Logs can be exported in JSON format for review and analysis by the teacher.

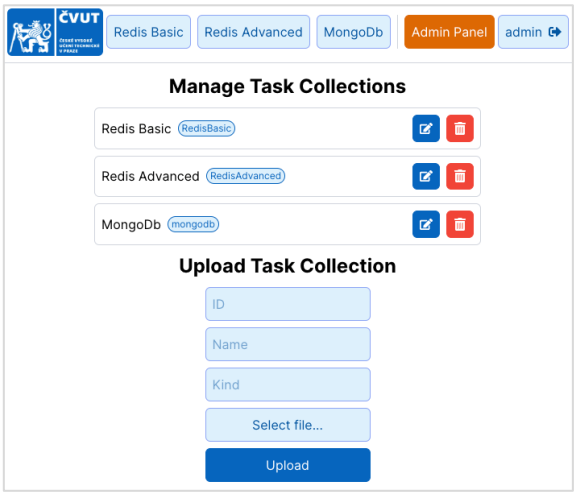


Figure 4: Screenshot of the instructor's panel (source: author's development).

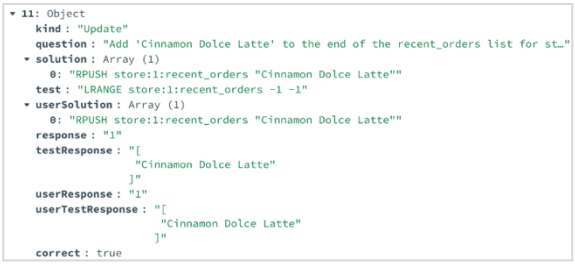
4 RESULTS AND DISCUSSION

For the pilot implementation of the system, 42 first-year master's students of the specialty "Open Informatics" were selected. All of them were willing to take the proposed tests. Participation in testing was not a mandatory element of the course: students took the tests to better prepare for the exam and to get additional feedback. This approach, on the one hand, ensures high motivation among test takers. Still, on the other hand, it may lead to a particular sampling bias (more interested or prepared students participate predominantly). In the future, it is planned to make the test mandatory and include it in the system of admission to the exam.

Students performed two types of tasks: first, they mastered basic operations (creating, selecting, updating, and deleting data in Redis) and then moved on to advanced assignments related to the RedisJSON and RediSearch modules. Each participant took the test individually, receiving basic and advanced tasks randomly.

The results showed that most students (about 78%) successfully coped with basic tasks on the first attempt. The main errors were related to syntactic inaccuracies in queries and attempts to scan across the hash fields. Thanks to the possibility of instant checking and resubmitting in case of a mistake, the average percentage of completed basic tasks reached 93% after two or three attempts. This indicates the system provides the dynamic feedback needed to identify and correct errors in Redis basic commands quickly.

Figure 5 shows a test log fragment with a student's successful answer.



▾ 11: Object
kind : "Update"
question : "Add 'Cinnamon Dolce Latte' to the end of the recent_orders list for st…"
▾ solution : Array (1)
0: "RPUSH store:1:recent_orders "Cinnamon Dolce Latte""
test : "LRANGE store:1:recent_orders -1 -1"
▾ userSolution : Array (1)
0: "RPUSH store:1:recent_orders "Cinnamon Dolce Latte""
response : "1"
testResponse : "[
"Cinnamon Dolce Latte"
]"
userResponse : "1"
userTestResponse : "[
"Cinnamon Dolce Latte"
]"
correct : true

Figure 5: Test log fragment (source: author's development).

A common issue arose when students were asked to create additional data structures (e.g., sets) to serve as reverse indexes linked to existing hashes. Although students demonstrated competence in using standard hash commands (HSET, HGET, etc.), they struggled with conceptualizing how to populate complementary sets for more advanced searching.

Students sometimes overlooked the specific naming conventions provided in the dataset. For instance, if the dataset used a particular prefix or a structured naming scheme (e.g., `store:1:equipment`), some students would create new keys without following the same format. This discrepancy could lead to mismatches with the reference solutions and to "incorrect result" flags in the automated checker.

Another issue was the misuse of spaces in key names. Students occasionally introduced unintended spaces or special characters, causing Redis to misinterpret the command and corrupt the intended data structure.

Another problem was the use of incorrect syntax when specifying the key "time to live". Many students forgot to provide all the necessary parameters or specified the time in the wrong units.

More problematic was the case with advanced Redis features, including RedisJSON and RediSearch capabilities. Only 39% of students could complete the test at the end of the first pass.

Many students struggled to configure the index correctly when using RediSearch for full-text search or indexing JSON documents. Some omitted essential fields, causing incomplete search results, while others declared fields with the wrong data type. Even if the index was set up correctly, students often encountered errors when making search queries, such as improperly accessing fields in a JSON document.

However, with repeated attempts and analysis of the errors displayed by the system, the success rate increased to 76%. This confirms that continuous automatic feedback gives students a deeper understanding of advanced Redis features.

In addition to quantitative measures for automatic assessment, the system provided valuable information for further adjustments to the course materials.

Thus, the results of implementation demonstrate the high efficiency of the system and the relevance of its use in the educational process. The proposed solution can be easily integrated into NoSQL training courses, complementing both classical lecture-practical formats and distance learning programs.

In the future, we plan to expand the assignment bank to include more "real-world" scenarios and to refine the hint system so that it not only indicates when an error has occurred but also gives guidance on how to correct it.

The advantage of the developed systém is that it is easily extensible to other NoSQL databases. In particular, it currently allows testing queries in MongoDB and can be extended to Cassandra and Neo4j. Such improvements will further enhance the quality of training and student satisfaction with the results of their studies.

5 CONCLUSIONS

A literature review shows that modern NoSQL technologies are widely recognized as necessary for academic curricula. However, automatic assessment, especially for Redis queries, remains poorly studied. Therefore, a system must be developed to evaluate Redis queries automatically.

The automated assessment system for student queries in Redis discussed in this paper demonstrates its effectiveness in teaching fundamental NoSQL concepts. Its key feature is instant feedback, which is provided in real-time. This allows students to identify and correct any syntax errors made immediately. Another essential capability of the created system is the support of advanced Redis modules.

Unlike existing online courses and sandboxes for Redis, the system presented here allows the instructor to customize assignment types (including advanced and more complex tasks), define a passing threshold, and track all user activity in real time.

An essential advantage of the system is its modular architecture, which allows scaling the solution for any scenario and expanding the task bank. In addition, the system is easily extensible to other NoSQL databases (MongoDB, Cassandra, Neo4j).

The pilot implementation showed that students have the most difficulties due to an incomplete understanding of the principles of working with key-value databases. For example, they try to perform cross-hash scanning across all fields in basic queries. In addition, they have difficulties mastering advanced Redis features related to RedisJSON and RediSearch modules. The high error rate in tasks related to index creation and full-text search highlights the need to emphasize these topics in both the course's theoretical instruction and practical classes.

Storing tasks and reference solutions in JSON format facilitates the automatic checking of results and subsequent analysis of student's practical skills. The possibility of flexibly adjusting the number and types of tasks and randomization mechanisms opens prospects for more accurately matching test sessions to individual students' skill levels. This, together, increases the efficiency of the educational process.

The study results confirm that automated query assessment systems are promising tools for university courses focused on modern data management technologies. They can be effectively applied in traditional lecture-practice learning formats and online learning platforms.

A further research direction could involve the development of structure-oriented comparison methods and flexible matching rules – such as case-insensitive checks, record-order independence and support for partial matches – to overcome the limitations of the current full-text comparison, broaden the system's applicability and yield more accurate assessments of student queries.

REFERENCES

[1] DB-Engines, "DBMS popularity broken down by database model," DB-Engines, [Online]. Available: https://db-engines.com/en/ranking_categories. [Accessed: 19-Feb-2025].

[2] N. Tripathi, "Teaching NoSQL databases in higher education: A review of models, tools and methods," SSRN, 2024, [Online]. Available: http://dx.doi.org/10.2139/ssrn.4971813.

[3] M. Menzin, S. Mohan, D. R. Musicant, and R. Soori Murthi, "NoSQL in undergrad courses is no problem," in Proceedings of the 51st ACM Technical Symposium on Computer Science Education, 2020, pp. 962-963, [Online]. Available: https://doi.org/10.1145/3328778.3366909.

[4] S. Mohan, "Teaching NoSQL databases to undergraduate students: a novel approach," in Proceedings of the 49th ACM Technical Symposium on Computer Science Education, 2018, pp. 314-319, [Online]. Available: https://doi.org/10.1145/3159450.3159554.

[5] M. Greiner, "Teaching NoSQL data models: a tutorial," in Proceedings of the 2021 AMCIS, 2021, Article 19.

[6] S. Kim, "Seamless integration of NoSQL class into the database curriculum," in Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education, 2020, pp. 314-320, [Online]. Available: https://doi.org/10.1145/3341525.3387399.

[7] M. Lin, J. Jun, Y. Zhu, and C. Zhan, "Research on the teaching reform of database curriculum major in computer in big data era," in Proc. 2017 12th Int. Conf. Computer Science and Education (ICCSE), IEEE, 2017, pp. 570-573.

[8] B. Fowler, J. Godin, and M. Geddy, "Teaching case: introduction to NoSQL in a traditional database course," Journal of Information Systems Education, vol. 27, no. 2, p. 99, 2016.

[9] A. Bajaj and W. Bick, "The rise of NoSQL systems: research and pedagogy," Journal of Database Management, vol. 31, no. 3, pp. 67-82, 2020, [Online]. Available: https://doi.org/10.4018/JDM.2020070104.

[10] C. Costa and M. Santos, "Big data: State-of-the-art concepts, techniques, technologies, modeling approaches and research challenges," IAENG International Journal of Computer Science, vol. 44, pp. 285-301, 2017.

[11] W. Wu, "Assessing peer correction of SQL and NoSQL queries," in Proceedings of the 54th ACM Technical Symposium on Computer Science Education V.1, 2023, pp. 535-541.

[12] A. Alawini, P. Rao, L. Zhou, L. Kang, and P.-C. Ho, "Teaching data models with TriQL," in Proceedings of the 1st International Workshop on Data Systems Education, 2022, pp. 16-21.

[13] Z. Li, S. Yang, K. Cunningham, and A. Alawini, "Assessing Student Learning Across Various Database Query Languages," 2023 IEEE Frontiers in Education Conference (FIE), College Station, TX, USA, 2023, pp. 1-9, [Online]. Available: https://doi.org/10.1109/FIE58773.2023.10343409.

[14] R. Alkhabaz, S. Poulsen, M. Chen, and A. Alawini, "Insights from student solutions to MongoDB homework problems," in Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V.1, 2021, pp. 276-282, [Online]. Available: https://doi.org/10.1145/3430665.3456308.

[15] M. Chen, S. Poulsen, R. Alkhabaz, and A. Alawini, "A quantitative analysis of student solutions to graph database problems," in Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V.1, 2021, pp. 283-289, [Online]. Available: https://doi.org/10.1145/3430665.3456314.

[16] A. Werner, "Applying NoSQL data adapter with the learning paths mechanism for better knowledge transfer in the age of distance learning," Procedia Computer Science, vol. 207, pp. 3330-3339, 2022, [Online]. Available: https://doi.org/10.1016/j.procs.2022.09.424.

[17] A. Werner and M. Bach, "NoSQL e-learning laboratory-interactive querying of MongoDB and CouchDB and their conversion to a relational database," in Communications in Computer and Information Science, vol. 865, 2018, pp. 581-592, [Online]. Available: https://doi.org/10.1007/978-3-319-67792-7_56.