Insights from Student Solutions to MongoDB Homework Problems

Ridha Alkhabaz, Seth Poulsen, Mei Chen, Abdussalam Alawini
{ridhama2,sethp3,meic2,alawini}@illinois.edu
University of Illinois at Urbana-Champaign

Abstract

We analyze submissions for homework assignments of 527 students in an upper-level database course offered at the University of Illinois at Urbana-Champaign. The ability to query databases is becoming a crucial skill for technology professionals and academics. Although we observe a large demand for teaching database skills, there is little research on database education. Also, despite the industry's continued demand for NoSQL databases, we have virtually no research on the matter of how students learn NoSQL databases, such as MongoDB. In this paper, we offer an in-depth analysis of errors committed by students working on MongoDB homework assignments over the course of two semesters. We show that as students use more advanced MongoDB operators, they make more Reference errors. Additionally, when students face a new functionality of MongoDB operators, such as `$group` operator, they usually take time to understand it but do not make the same errors again in later problems. Finally, our analysis suggests that students struggle with advanced concepts for a comparable amount of time. Our results suggest that instructors should allocate more time and effort for the discussed topics in our paper.

CCS Concepts

• **Applied computing** → **Education**; • **Social and professional topics** → **Computer science education**; • **Information systems** → **Information retrieval**; **Query representation**.

Keywords

mongoDB, database education, online assessment

ACM Reference Format:
Ridha Alkhabaz, Seth Poulsen, Mei Chen, Abdussalam Alawini. 2021. Insights from Student Solutions to MongoDB Homework Problems. In *26th ACM Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2021), June 26-July 1, 2021, Virtual Event, Germany.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3430665.3456308

1 Introduction

The evolution of big data exposed the limitations of relational (SQL) databases. Internet giants, such as Google and Amazon, developed and used many custom-built databases to work around the shortcomings of SQL databases. MongoDB, a cross-platform document-oriented database that uses JSON-like documents, was introduced in 2007 as part of this NoSQL (Not Only SQL) revolution. Since its inception, MongoDB has gained a great reputation and has become one of the most used and wanted database systems [16].

The flexibility of MongoDB's semi-structured documents and its ability to effectively scale to distributed collections of documents made several large companies shift from SQL to MongoDB. MongoDB provides higher scalability and availability than its SQL-based counterparts [8]. It also allows users to develop user-defined functions written in JavaScript as well as write Map-Reduce programs. Map-reduce infrastructure enables users to write distributed aggregate computations over large, highly-distributed, volumes of data [7].

In response to the new interest in NoSQL, many universities and educational organization have adopted curriculum that includes NoSQL database management systems [3, 5, 6, 12, 13].

Addressing the growing interest of students and instructors in MongoDB, in this paper, we give insight about students' experience when learning MongoDB using a quantitative approach. We classified 76,168 submissions made by 527 students over Fall 2019 and Spring 2020 semesters while working on homework problems at the University of Illinois at Urbana-Champaign. This contributes to the growing literature about analyzing and understanding students' mistakes when learning a new database language. In this paper, We answer the following research questions: (1) *What concepts are difficult for students when learning MongoDB?* (2) *What common errors do students make when first learning to query a MongoDB database?*

2 Literature Review

Understanding novice programming behavior has been one of the most researched topics of Computing Education research [18]. More specifically, educators wanted to understand the effectiveness of CS1 courses and material on novices' knowledge attainment and the ability to deploy their gained programming skill. Consequently, many studies tried to find common struggles between novices attending CS1 courses. For instance, Lahtinen et al. (2005) reported that "dividing functionality into procedures," and "finding bugs from their own programs" [11] were one of the most expressed struggles among novices in CS1 courses. Furthermore, other studies showed that problems related to loops, arrays and passing data to/from modules were more frequent than others [18]. The previous studies among many others exhibit students' perceived mental models faults in depicting programming mechanism.

However, unlike object-oriented and imperative programming, there is not a lot of research about teaching students to program in database systems and query languages, and the little existing research is mostly limited to SQL. One exception is the analysis by Chen et al. on the errors that students make while learning

This work is licensed under a Creative Commons Attribution International 4.0 License.

ITiCSE 2021, June 26-July 1, 2021, Virtual Event, Germany.
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8214-4/21/06.
https://doi.org/10.1145/3430665.3456308

to write graph database queries [4]. Though there are no existing studies on the difficulties encountered by students while learning to use MongoDB, there are multiple reports of instructors including MongoDB into their curriculum in university courses [3, 5, 6, 12, 13].

Guo et al. integrated MongoDB into security labs [6]. These efforts mainly focus on how to order topics related to database security and how to introduce some of the security drawbacks of MongoDB's implementation. Lei et al. [12], use MongoDB as their NoSQL database management system in their labs to teach the drawbacks of NoSQL databases. Mohan reported experiences of a database education curriculum that incorporated NoSQL [13]. In Mohan's work, students were exposed to several NoSQL paradigms and had a set of projects, lab and research assignments to complete using the knowledge they gained during the course. The course received positive feedback from students and industry big data engineers for its addressing of NoSQL paradigms. Some other NoSQL databases have also been incoporated in university curricula. For example, Fowler et al. reported their experience in two database courses with teaching CouchDB, a NoSQL data management system that uses JavaScript as its query language [5]. They mainly focused on measuring students' improvement of understanding NoSQL systems. Although. the previous papers include MongoDB in their studies, but they do not study novices' experiences or difficulties when learning MongoDB.

Previous work in SQL education has looked at thousands of student submissions to both homework and tests to understand what types of queries are most difficult for students to write, and what types of errors students make the most [1, 2, 17, 19]. In this paper, we follow their lead, giving a quantitative insight from students' solutions of MongoDB homework questions in order to find common difficulties among novices.

3 Introduction to MongoDB

MongoDB [9] is a document-oriented NoSQL database that stores JSON-like data in documents with dynamic schemata, so it can store flexible values without fixing the number of fields or type of fields. We will show examples of MongoDB objects and some code snippets that exhibits the basic syntax of MongoDB and the corresponding queries in SQL with similar database schema. The '_id' attribute is an indexed attribute that every object must have, and it acts as the object's primary key. We will show two instances of objects that might appear in collection titled 'Movies', to demonstrate the flexibility in MongoDB, avoiding SQL's rigid relational structure and the need to translate objects to relational tables. One object in such a collection might look like:

```
{
    "movie_id": 1,
    "movie_name": "The Imitation Game",
    "release_year": 2014,
    "country": "USA",
    "director": "Morten Tyldum"
}
```

And another object in the 'Movies' collection that doesn't have the information about director but has the attribute 'ratings', might be:

```
{
    "movie_id": 2,
    "movie_name": "The Social Network",
    "release_year": 2010,
    "country": "USA",
    "ratings": 7.7
}
```

Observing the instances above, though the structures of the items in a collections are not the same, MongoDB deals with these kinds of collections flexibly. Compared to SQL relational tables and rigid structure, MongoDB databases have the potential for scalability because of its flexibility when dealing with collections.

MongoDB databases can be queried from many different programming languages, but the most straightforward interface is querying it through a JavaScript shell, so that is what the students are taught in our class. The following is an example of a simple MongoDB query, written in JavaScript shell, to find the movie ids of all horror movies in our example database.

```
db.Movies.find(
    { movie_genre: "Horror" },
    { _id: 0, movie_id: 1, movie_genre: 1}
)
```

The following code is the corresponding SQL syntax with an analogous database that has similar attributes in the database.

```
SELECT movie_id, movie_genre
FROM Movies
WHERE movie_genre = "Horror"
```

More complex operations on data in MongoDB are done with MongoDB operators, which start with a '$' character [10]. For example, $match is an operator that takes a specified conditions as its inputs in order to filter and produce documents that have met set conditions. Another example, $unwind is an aggregation operator, which takes a reference to an array and produces multiple objects from the elements of the array. Another aggregation operator is $group, which takes an _id field as its first argument, then, it takes fields combined with accumulator operators to perform basic operations, such as $sum, on the collection. Finally, $project is an operator that takes fields as its arguments, then adds, renames, excludes or includes the specified fields in the resulting collection.

Having the structure of the previous collection in mind, we will now show an example that uses the previous aggregation operators to produce a collection. This query finds all movies in the collection 2001 or after, calculates their average ratings, and renames the country field to produce_country:

```
db.Movies.aggregate([
    {$match: {release_year: {$gte:2001}}},
    {$unwind: "$country"},
    {$group: {
        _id: "$country",
        ave_ratings: {$avg:"$ratings"}} },
    {$project: {
        ave_ratings:1,
        "produce_country": "$_id",_id:0 } }
])
```

The following SQL query functions comparably to the above MongoDB query on an analogous database with similar attributes.

```
SELECT AVG(ratings) as ave_ratings,
       country as produce_country
FROM Movies
WHERE release_year >= 2001
```

4 Data & Methods

The University of Illinois at Urbana-Champaign (UIUC) is a research-intensive institution with about 1,800 undergraduate Computer Science majors. The data was collected from the Database Systems course at UIUC [3]. The Database Systems course (CS411) is an elective course taken primarily by graduate students or undergraduates nearing the end of their degree, with pre-requisites including introduction to programming and data structures. CS411 is structured to cover four main units: *data models, database management systems and query languages* (relational model: relational algebra, SQL and MySQL, graph model: Neo4j and cypher and document-oriented model: JavaScript shells and MongoDB), *database design* (conceptual design and normal forms) and *database implementation* (storage and indexing, query optimization, concurrency control). The course spends three lectures to introduce MongoDB. Our data contains submissions from Fall 2019 and Spring 2020 from students in CS411. Our sample of students was 527 students, including 64 female and 463 male students.

The students in CS411 completed their homework using PrairieLearn, an online homework and exam platform [21]. Upon submitting a MongoDB query in PrairieLearn, the students were given instant feedback from auto-graders. Students had unlimited number of attempts and no constraint in what order they may approach the questions. After close inspection of the statistics we found that the trends in student errors and number of submissions were similar for both semesters included in the study, so we will report them in aggregate.

For each MongoDB problem, the students were given a problem prompt and a general description of the collections related to the problem. Students wrote their MongoDB queries in a JavaScript MongoDB enabled shell, where they used JavaScript as a query language.

Student had small text editor where they wrote their queries and had the option of saving and grading them or just saving them. When the student chooses to grade their submission, the query run against the MongoDB objects. Should the query face a JavaScript error or a MongoDB error, it would be reported back to the student with a descriptive message from the shell. If the query successfully runs, the student can see whether they have an incorrect result set (did not receive full points) or whether they achieved the desired output (received full points).

Also, it is worth mentioning that 2,715 students' submissions were omitted from this study, because they had a common error where students copied JavaScript code from online sources, causing the interpreter to fail due to an unexpected unicode character. In addition, we do not include question ten of the homework in our analysis, because it was optional and not many students attempted to solve it.

4.1 Data Handling

All graduate and undergraduate researchers completed training in responsible conduct of research. To protect students' privacy, data was anonymized before being accessed by student research assistants.

4.2 Categorization

Prior work in understanding student struggles while writing database queries with SQL have partitioned errors into different categories based on the type of error returned by the SQL engine [2, 17, 20]. However unlike most SQL engines, which will only return a single error message to the user at a time, JavaScript shells, which are used in executing MongoDB queries, may report any number of errors when code is run. This significantly complicates the process of classifying student errors. In order to simplify the process of arriving at a meaningful categorization, we chose to classify each submission using only the first error which was returned. Anecdotally, we find that students often only focus on fixing one error at a time. Furthermore, often one JavaScript error leads to another in such a way that fixing the first error resolves all of them (for example, a Syntax error on a line that defines a variable will cause an Undefined reference error on the line that variable is used). Thus, we feel that examining only the first error is a valid approach to examining student mistakes.

Another aspect which complicates the categorization of MongoDB errors is the fact that MongoDB queries are written through JavaScript library, rather than having its own dedicated language like SQL or Neo4j's Cypher query language [14, 15]. In practice, this means that code written to query a MongoDB database could fail on a JavaScript error before the MongoDB library code is even invoked, or the JavaScript could run successfully, only to have the MongoDB library return an error saying that though it is valid Javascript, it does not comply with the particular usage rules of the library.

Thus we arrive at a way to partition student submissions into 4 categories: *JavaScript Errors*, where the first error returned by the JavaScript library comes from a JavaScript failure, rather than from within the MongoDB library, *MongoDB errors*, where first error is an error message given by MongoDB library denoting that the query is incorrect, *Incorrect result set*, where the query is valid and is executed by MongoDB, but returns an incorrect result, and *correct solutions*, where the query is executed by MongoDB, and the returned result set matches the expected result set.

4.3 Overview of Homework Assignments

The ten homework questions are designed to address the following concepts, and students were free to move between the problems however they wanted:

(1) Selection and projection using `find()` (Basic): find document(s) that satisfies certain conditions. (1 question)
(2) Selection and projection using `find()` (Advanced): find document(s) that satisfies complex conditions. (1 question)
(3) Aggregation pipeline with group and project: aggregate values of certain keys in documents. (1 question)
(4) Aggregation pipeline with match, sort, group and project: filter, sort and aggregate values of certain keys in documents. (1 question)
(5) Aggregation pipeline with unwind operator: dis-aggregate array elem into document of the array's items. (1 question)
(6) Querying arrays: finding documents with certain array elements. (1 question)

(7) Querying embedded documents: query documents embedded inside other documents. (1 question)
(8) Querying linked documents using cursor: Use cursor methods to iterate over documents and perform certain tasks. (1 question)
(9) Map Reduce: Use the map-reduce platform to perform grouping and aggregation over distributed documents (1 question)
(10) Extra Credit: Map Reduce (Advanced) (1 question)

4.4 Data Overview

We will now show an example of one of the homework problems and student solutions to that problem to demonstrate the information that is available to us in our data set. We will be looking at a student's work while trying to solve the following homework problem (the question from Spring 2020 addressing topic 3):

> *Given a MongoDB database with two collections,* ***Movies*** *and* ***Actors****. For each movie genre, display the genre and the average ratings of movies under that genre. Your query should only output the name of the genre (rename it as "movie_genre") and average ratings of movies (output it as "ave_ratings").*
>
> Database Description:
> **Movies** collection: each Movie has its unique id ($movie_id), name ($movie_name), country ($country), director ($director), releasing date ($release_year), ratings ($ratings), genre ($genre), and $actors [an array of actor id($actor_id)]. Every movie is associated with only one genre and one director.
>
> **Actors** collection: each actor has his/her unique id ($actor_id), name ($actor_name), and his/her birth country ($birth_country).

We will follow one student's solution sequence as a way of exploring the data set. They first attemepted to answer the prompt with the following query:

```
db.Movies.aggregate({
    $group: {
        _id: "$genre",
        ave_ratings: { $avg : "$ratings"}
    }
})
```

And they received the following feedback message:

```
Expected results
===============
{
    "movie_genre": "Horror",
    "ave_ratings": 5.869565217391305
}

Actual results
==============
{
    "_id": "Horror",
    "ave_ratings": 5.869565217391305
}
```

The problem here was the student did not use the $project operator in order to rename the _id field as movie_genre. Then, the student came up with this MongoDB query in order to solve the renaming issue:

```
db.Movies.aggregate({
    $group: {
        movie_genre : "$genre",
        ave_ratings : { $avg : "$ratings"}
    }
})
```

but then they received the following error message:

```
Error: command failed: {
    errmsg : "The field 'movie_genre' must be an
        accumulator object",
    code : 40234,
    codeName : "Location40234"
}
```

The problem here was the misuse of the $group operator, which takes the first argument as a field name to group the resulting set by, then takes the following arguments to calculate the aggregate operations on the specified fields. This is a very common problem when students try to learn the functionality of $group. Table 4 shows that students often struggled with this error when solving this particular question.

After a few more tries, the student submitted the below MongoDB query using the $project operator to rename _id as movie_genre. The $project operator takes the fields' names, and parameter that either excludes (:0) or includes (:1) the field in the resulting document(s). It can also be used to rename a field in the output.

Finally, the student was able to get the correct answer:

```
db.Movies.aggregate({
    $group: {
        _id : "$genre",
        ave_ratings : { $avg :"$ratings" }
    }
}, {
    $project: {
        movie_genre: "$_id",
        ave_ratings : 1,
        _id: 0
    }
})
```

5 Results

Table 1 shows the breakdown of our categorization in the left column, and the percentages of each main category with respect to all submissions in the right column. In this table, we see that the majority of submissions are classified under "Incorrect result set", which accounts for 47% of all submissions. Submission result in "Incorrect result set" for a variety of reasons. Section 4.4 includes one of the students' MongoDB queries that produced an incorrect result set by not incorporating $project in their MongoDB query. Due to the complexity of understanding the variety of "Incorrect result set" submissions, we leave further analysis of these errors to future work, and will focus mostly on JavaScript errors and MongoDB errors here.

Tables 2 and 3 show the most common JavaScript and MongoDB errors in their right column and their percentages of appearing among MongoDB or JavaScript errors. As shown in the first row

Category	Percentage
(1) Incorrect result set	47 %
(2) MongoDB error	32 %
(3) JavaScript error	6 %
(4) Correct	14 %

Table 1: This is the General breakdown of our Categorization

JavaScript Error Codes	Percentages
(1) Syntax error	68 %
(2) Type error	27 %
(3) Failed to parse	3 %
(4) Unknown error	2 %

Table 2: Breakdown of JavaScript error messages percentages

MongoDB Error Codes	Percentages
(1) Reference error	**60 %**
(2) Field must be an accumulator object	8 %
(3) Unknown operator $and	6 %
(4) Undefined field	4 %
(5) A pipeline stage specification object must contain exactly one field	4 %
(6) Unrecognized pipeline stage name: _id $	3 %
(7) Unknown group operator	2 %
(8) Cannot return an array from Map Reduce	1 %
(9) Assert failed	1 %
(10) Namespace not found	1 %
(11) Unrecognized expression	1 %
(12) An object representing an expression must have exactly one field	1 %
(13) Illegal number of arguments used in `$gte`.	1 %
(14) Fields' name shouldn't begin with '$'	1 %
(15) Path option to `$unwind` stage should be prefixed with a '$'	1 %
(16) Bad projection specification, excluding more fields than required	1 %

Table 3: Breakdown of MongoDB error messages percentagewise

of Table 3, the most prominent MongoDB error is Reference error, which happens when students do not address the specific fields correctly. Hence, Reference errors have multiple causes, such as misspelling a field name or misunderstanding the structure of the data output by a MongoDB operator. Furthermore, Table 4 fifth column, in particular questions 5 to 8, shows that on advanced MongoDB concepts, students have more Reference errors. Likewise, the authors previous work shows similar students' performance trends between advanced SQL queries and Undefined Column errors [17].

Table 3 reveals some problems students face when writing MongoDB queries' operators against a certain fields or operators with or without prefixing $, specifically seen in rows 6, 14 and 15. Table 2 third entry shows students' problems with parsing their MongoDB queries in the JavaScript shell. Table 4 third column indicates that Type errors spike in question 3. Type errors occur when students do not pass the correct arguments to a MongoDB operator. Therefore, since question 3 is the first question that assesses aggregate operators, such as `$group` and `$project`, students spend more time and effort getting accustomed to the dynamics of aggregate operators.

Table 5 shows the distribution of students' individual submissions per question by median time (second column), the numbers of submissions (third column), the number of students who attempted the questions (fourth column) and the number of students who completed the question (fifth column). Table 5 fourth and fifth columns show us that most students completed each required question. Also, Table 5 confirms that students take more time when introduced to a new MongoDB operators, as we notice that median time of sessions positively correlate with the difficulty of the questions. The median time is calculated from our median of students' session duration sample for each question, where we calculate sessions using the time elapsed from the first submission and the last submission in a sequence where the time difference between two submissions is not longer than 15 minutes.

6 Discussion

In section 5, we mentioned that as students get assessed in new MongoDB operators, they make more mistakes and need more time to generate the expected dataset. This argument finds further validity in Table 4. The fifth column of Table 4 indicates that advanced MongoDB queries and operators had a comparable and high number of Reference errors.

Another student behavior we found interesting is that when students are assessed in a new MongoDB operator, they spend more time to learn and tend to make less of that mistake as time progress. Table 4 sixth column shows that students make the highest number of *Field must be an accumulator object* in question 3, which is the question in which the `$group` operator is first assessed. Considering that the `$group` operator was also assessed in most of the remaining questions (i.e., questions 4-8), the *Field must be an accumulator error* appear significantly less in the later questions since students have already solved question 3 and are now more comfortable with the `$group` operator.

In addition, we observe in Table 5 that students spend comparable time when approaching advanced concepts in MongoDB, specifically in questions 5 through 9. Also, Table 4 second and fifth columns show that students made comparable amount of reference errors and syntax errors in questions 5 to 9.

6.1 Limitations

For the sake of simplicity we only considered the first appearing error message for the student. A thorough examination of all error messages may give more in-depth insight into novices' experience with MongoDB. Also, our study only considered queries as individual entities, so, considering the context of submissions, including what error messages students received before and after the current submission, can reveal more details about students' difficulties when learning MongoDB.

Our data set only had two problem statements for each concept. Hence, we couldn't give insight about the effect of problems'

	Javascript Errors				MongoDB Errors		
	Syntax error	Type error	Failed to parse	Unknown error	Reference error	Field must be an accumulator object	Unknown operator $and
(1) Selection and projection using `find()` (Basic)	108	46	0	0	890	9	145
(2) Selection and projection using `find()` (Advanced)	**672**	34	3	0	1774	14	12
(3) Aggregation pipeline with group and project	372	116	76	0	854	**1250**	7
(4) Aggregation pipeline with match, group, sort and project	353	**473**	19	0	768	137	244
(5) Aggregation pipeline with unwind operator	343	68	9	17	**1873**	150	213
(6) Querying arrays	402	122	6	0	**2395**	251	284
(7) Querying embedded documents	496	42	1	0	**2413**	42	390
(8) Querying linked documents using cursor	428	216	46	0	**2150**	100	146
(9) Map Reduce	292	228	0	87	1369	4	42

Table 4: Breakdown of most common Javascript and MongoDB errors by question

Question	Median Time (MM:SS)	# Submissions	# Attempted Questions	# Completed Questions
(1) Selection and projection using `find()` (Basic)	11:00	2788	527	525
(2) Selection and projection using `find()` (Advanced)	42:00	6563	524	524
(3) Aggregation pipeline with group and project	42:00	7899	525	525
(4) Aggregation pipeline with match, group, sort and project	42:03	7421	523	522
(5) Aggregation pipeline with unwind operator	43:00	10014	521	518
(6) Querying arrays	32:14	10951	521	517
(7) Querying embedded documents	38:59	10528	517	515
(8) Querying linked documents using cursor	53:00	9811	514	510
(9) Map Reduce	51:00	10216	502	499

Table 5: The Breakdown of Concepts and attempts to apply them with median time

wording on students' understanding. In addition, most of students completed the question prompts and students had a plenty of time to do the homework. This made it impossible to gauge the difficulty of the questions by looking at completion rates, due to strong ceiling effects. In addition, our data did not include students' ethnic or cultural background.

Another limitation is that we examine only the error output of the queries, not the queries themselves. We could gain greater insight into why students obtained so many "Incorrect result set" errors by doing a qualitative analysis of the code that the students wrote. Future work could also benefit greatly from talk-aloud interviews with students in order to gain better understanding of student's though processes while writing database queries.

7 Conclusion

In this paper we shed a light on students experiences when learning a NoSQL Database, MongoDB, using over 76 thousands MongoDB queries written by 527 students. Our work shows that students take comparable time and number of submissions on advanced concepts. Also, we found that students make more Reference errors in advanced queries than other queries. In addition, we found that students seem to make more JavaScript errors when they write more complex MongoDB queries. These results should help instructors to enhance their curriculum by allotting students more time for the MongoDB concepts discussed above and giving novices better expectations about learning MongoDB.

Acknowledgments

This work supported in part by NSF IUSE grant 2021499. We would also like to thank emeritus professor Michael C. Loui for his thoughtful notes and feedback.

References

[1] Alireza Ahadi, Vahid Behbood, Arto Vihavainen, Julia Prior, and Raymond Lister. 2016. Students' Syntactic Mistakes in Writing Seven Different Types of SQL Queries and Its Application to Predicting Students' Success. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16)*. ACM, New York, NY, USA, 401–406. https://doi.org/10.1145/2839509.2844640

[2] Alireza Ahadi, Julia Prior, Vahid Behbood, and Raymond Lister. 2015. A Quantitative Study of the Relative Difficulty for Novices of Writing Seven Different Types of SQL Queries. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (Vilnius, Lithuania) *(ITiCSE '15)*. ACM, New York, NY, USA, 201–206. https://doi.org/10.1145/2729094.2742620

[3] Abdussalam Alawini. 2021. Overview - CS411 Database Systems Spring 2021 - Illinois Wiki. https://wiki.illinois.edu/wiki/display/CS411AASP21

[4] Mei Chen, Seth Poulsen, Ridha Alkhabaz, and Abdussalam Alawini. 2021. A Quantitative Analysis of Student Solutions to Graph Database Problems. In *Proceedings of the 2021 ACM Conference on Innovation and Technology in Computer Science Education* (Virtual Event, Germany) *(ITiCSE '21)*. Association for Computing Machinery, New York, NY, USA, 7.

[5] Brad Fowler, Joy Godin, and Margaret Geddy. 2016. Teaching case: introduction to NoSQL in a traditional database course. *Journal of Information Systems Education* 27, 2 (2016), 99.

[6] Minzhe Guo, Kai Qian, and Li Yang. 2016. Hands-on labs for learning mobile and NoSQL database security. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 2. IEEE, 606–607.

[7] MongoDB Inc. 2020. Map-Reduce. https://docs.mongodb.com/manual/core/map-reduce/

[8] MongoDB Inc. 2020. MongoDB vs MySQL. https://www.mongodb.com/compare/mongodb-mysql

[9] MongoDB Inc. 2020. The most popular database for modern apps. https://www.mongodb.com/

[10] MongoDB Inc. 2020. Operators. https://docs.mongodb.com/manual/reference/operator/

[11] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. A Study of the Difficulties of Novice Programmers. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Caparica, Portugal) *(ITiCSE '05)*. Association for Computing Machinery, New York, NY, USA, 14–18. https://doi.org/10.1145/1067445.1067453

[12] Lei Li, Kai Qian, Qian Chen, Ragib Hasan, and Guifeng Shao. 2016. Developing Hands-on Labware for Emerging Database Security. In *Proceedings of the 17th Annual Conference on Information Technology Education* (Boston, Massachusetts, USA) *(SIGITE '16)*. Association for Computing Machinery, New York, NY, USA, 60–64. https://doi.org/10.1145/2978192.2978225

[13] Sriram Mohan. 2018. Teaching NoSQL Databases to Undergraduate Students: A Novel Approach. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 314–319. https://doi.org/10.1145/3159450.3159554

[14] Neo4j Inc. 2019. Neo4j. https://neo4j.com/

[15] Oracle Corporation. 2019. MySQL. https://www.mysql.com/

[16] Stack Overflow. 2019. Stack Overflow Developer Survey 2019. https://insights.stackoverflow.com/survey/2019/ [Online; accessed 10-January-2020].

[17] Seth Poulsen, Liia Butler, Abdussalam Alawini, and Geoffrey L. Herman. 2020. Insights from Student Solutions to SQL Homework Problems. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE '20)*. Association for Computing Machinery, New York, NY, USA, 404–410. https://doi.org/10.1145/3341525.3387391

[18] Anthony V. Robins. 2019. *Novice Programmers and Introductory Programming*. Cambridge University Press, 327–376. https://doi.org/10.1017/9781108654555.013

[19] Toni Taipalus and Piia Perälä. 2019. What to Expect and What to Focus on in SQL Query Teaching. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 198–203. https://doi.org/10.1145/3287324.3287359

[20] Toni Taipalus, Mikko Siponen, and Tero Vartiainen. 2018. Errors and Complications in SQL Query Formulation. *ACM Trans. Comput. Educ.* 18, 3, Article 15 (Aug. 2018), 29 pages. https://doi.org/10.1145/3231712

[21] Matthew West, Geoffrey L. Herman, and Craig Zilles. 2015. PrairieLearn: Mastery-based Online Problem Solving with Adaptive Scoring and Recommendations Driven by Machine Learning. In *2015 ASEE Annual Conference & Exposition*. ASEE Conferences, Seattle, Washington, 26.1238.1–26.1238.14. https://peer.asee.org/24575.