A Quantitative Analysis of Student Solutions to Graph Database Problems

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

Abstract

As data grow both in size and in connectivity, the interest to use graph databases in the industry has been proliferating. However, there has been little research on graph database education. In response to the need to introduce college students to graph databases, this paper is the first to analyze students' errors in homework submissions of queries written in Cypher, the query language for Neo4j—the most prominent graph database. Based on 40,093 student submissions from homework assignments in an upper-level computer science database course at one university, this paper provides a quantitative analysis of students' learning when solving graph database problems. The data shows that students struggle the most to correctly use Cypher's `WITH` clause to define variable names before referencing in the `WHERE` clause and these errors persist over multiple homework problems requiring the same techniques, and we suggest a further improvement on the classification of syntactic errors.

CCS Concepts

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

Keywords

Neo4j, database education, online assessment

ACM Reference Format:
Mei Chen, Seth Poulsen, Ridha Alkhabaz, Abdussalam Alawini. 2021. A Quantitative Analysis of Student Solutions to Graph Database 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/10.1145/3430665.3456314

1 Introduction

The relational database has been used for decades [7], but as the amount of data and the need to store data that is rich in relationships is increasing drastically, a special kind of the NoSQL database model has emerged: graph databases [17]. Graph databases store relationships and connections based on the fundamental graph theory constructed via nodes (entities) and edges (relations), making it an optimal choice to store and query graph structures and seeking to provide both better performance and better usability for the right kinds of data. In 2017, over half of the enterprise users across all industries utilize graph database due to its speed and enhanced execution [26].

In this paper, we focus on the Neo4j database and its Cypher language, a declarative query language that has SQL-like syntax augmented by the ability to pattern match on graph relationships. Neo4j is the most popular graph database: eBay uses Neo4j to support the probabilistic models and aid understandings in the conversational shopping scenario; NASA uses Neo4j to help examine the relationship between knowledge; more companies [22] have started using graph model thanks to its capability in constructing relationships and its simplicity in both visual understandings and agility in processing graph-related data.

The power of Neo4j is also supported by academic studies. According to Fernandes and Bernardino, Neo4j stands out among the current graph database thanks to its simplicity, agility, and flexibility [10]. Vicknair et al. also used Neo4j to demonstrate that the graph system shows better performance in full-text character searches and structural type queries compared to that in the relational database [28].

The need for introducing students to graph databases is also rising with potential pervasive usage in multiple areas such as chemistry, social networking, and recommendation engines [6, 20]. Many universities have begun covering Neo4j in their course, including but not limited to: Portland State University, the University of Pennsylvania, the University of Illinois at Urbana-Champaign, and ETH Zürich [3, 8, 11, 27]. Additionally, Coursera, the online learning platform, offers introductory Neo4j courses [9, 15].

In response to the need for graph database talents and the lack of CS education research on how students learn about graph database, we analyze thousands of students submissions to Graph Database assignments presented in an upper-lever database course offered at a large public school. In particular, we study those research questions: (1) What is the distribution of correct submissions, semantic errors, and syntactic errors for students writing Neo4j queries? (2) What common errors do students make when they first use the Cypher query language? and (3) Which concepts students spend most time learning?

2 Literature Review

Relational database management systems (RDBMS) have been long-established and used in industries for decades, but due to the rise of data that are bigger both in size and in interconnectivity, there is a trend to utilizing non-relational databases, as they are more efficient for certain use cases [18, 24].

However, computer science education research on NoSQL database languages is significantly lacking compared to that on SQL databases. Ahadi et al. provide a quantitative analysis of relative difficulties in SQL queries [1, 2]. The authors' previous work uses quantitative approaches to identify the most common students'

ITiCSE 2021, June 26-July 1, 2021, Virtual Event, Germany

ACM ISBN 978-1-4503-6874-2/20/06.
https://doi.org/10.1145/3430665.3456314

errors when writing SQL queries [23]; other papers study the potential difficulties students face from an ease-of-use perspective [25]. Some researchers have proposed novel tutors and tools to help with teaching SQL [5]. There are rare studies conducted about the NoSQL database. Fowler demonstrated a successful teaching case that incorporated the NoSQL database into the traditional database management course to students and received a significant improvement in understandings NoSQL [12]. Sriram proposed a four-tiered learning model to better teach NoSQL Databases to undergraduate students [21]. Alkhabaz et al. provide an analysis of the types of errors students make while learning to write MongoDB queries [4].

We have only found one study about graph database in computer science education [16] that conducted a Neo4j teaching case that shows the successful improvements in Vocational Education and Training environment. Other researches on graph databases are mainly about performance analysis on different operating systems or introductions on the differences between graph database and RDBMS or other NoSQL database [13, 28].

To the best of our knowledge, there has been no research in understanding the mistakes students make while learning query graph databases. In this paper, we show the descriptive statistics derived from the students' submissions in homework problems and investigate what concepts students struggle with most. To that end, we look at students' median time to finish and the number of submissions.

3 Introduction to Neo4j and Cypher

Cypher is the query language used to query Neo4j databases. It has a similar syntax to SQL, with declarative pattern-matching features added for querying graph relationships. As one of the best-known graph databases, Neo4j stands out among the current existing graph models for its performance, simplicity and its powerful query language [10, 14].

In this section we briefly demonstrate the syntax of Neo4j queries, showing the similarities and differences when compared to SQL via two analogous databases. The graph database we will use for our examples only has two kinds of nodes: `Movie` and `Actor`, and one relationship: `ACTED_IN` between the two kinds of nodes.

> **Movie** node: each Movie has its unique id (`movie_id`), its name (`movie_name`), release year (`release_year`), ratings (`ratings`), genre (`genre`).
>
> **Actor** node: each star has their unique id (`actor_id`), name (`actor_name`), birth year (`birth_year`) and their birth country (`birth_country`).
>
> Actor and Movie nodes have the relationship: `(:Actor)-[:ACTED_IN]->(:Movie)`.
> An actor may act in many movies and a movie may have many actors.

Figure 1 shows a graph representation of some example data that may be present in a database with the defined schema.

The equivalent relational database we use for this example has three tables:

> **Actor**: `actor_id` (INT), `actor_name` (VARCHAR), `birth_year` (INT), `birth_county` (VARCHAR)
> **ACTED_IN**: `actor_id` (INT), `movie_id` (INT)

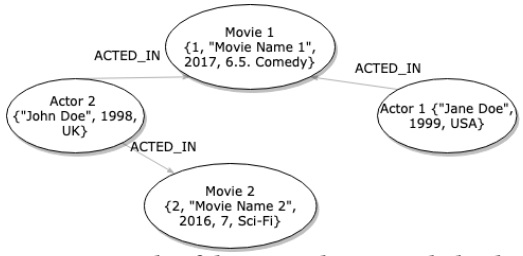

Figure 1: Example of data stored in a graph database.

> **Movie**: `movie_id` (INT), `movie_name` (VARCHAR), `release_year` (INT), `ratings` (REAL), `genre` (VARCHAR)

In order to find the birth country of an actress named "Jane Doe", we would use the following Cypher query:

```
MATCH (a:Actor)
WHERE a.actor_name = "Jane Doe"
RETURN a.actor_name, a.birth_country
```

The corresponding SQL code is very similar to the Neo4j query with only slight difference in naming of the keywords:

```
SELECT a.actor_name, a.birth_country
FROM Actor AS a
WHERE a.actor_name = "Jane Doe"
```

Now that we have shown a simple example, we will examine an example that shows the strength of Neo4j in dealing with data that has a graph structure. The following query will find the number of comedy movies that each actor has acted in:

```
MATCH (m:Movie {genre:"Comedy"})-[:ACTED_IN]-(a:Actor)
RETURN a.actor_name, COUNT(m.movie_id) AS count_movie
```

Here we can see the relationship of actors and movies in the Neo4j. Compared to the SQL query, Cypher does not require any `JOIN` or `GROUP BY` keywords, and only use pattern matching to find actors who acted in a "Comedy" movie. On the other hand, the equivalent SQL query is more complex as it requires joining three tables and grouping by the actor name:

```
SELECT a.actor_name, COUNT(m.movie_id) AS count_movie
FROM Actor AS a
JOIN Acted_IN AS act ON (a.actor_name = act.actor_name)
JOIN Movie AS m ON (act.movie_id = m.movie_id)
WHERE m.genre = "Comedy"
GROUP BY a.actor_name
```

4 Methods

In this section, we first describe the data collection and handling process, and then we present an example of a student solving a homework problem. Finally, we discuss how we categorized students' submissions.

4.1 Data Collection

The data was collected by parsing homework submissions made by students taking the database systems course at the University of Illinois at Urbana-Champaign [3]. We analyze data from student submissions from the Fall 2019 and Spring 2020 offerings of a database course, which is divided into four modules: (1) *data models*

and query languages, (2) *relational database design*, (3) *relational database system internals*, and (4) *advanced database topics*

In the first module, students learn about SQL, MongoDB, and Neo4j sequentially. They learn about label property graph data model and the Cypher query language (Neo4j) in two class meetings, and they have two activities (5 problems each) to finish in class and one homework (10 problems) to finish after the classes.

Students submitted their assignments on PraireLearn, an online homework and exam platform which automatically collects all the submission information [29]. The students are allowed an unlimited number of submissions for homework problems without penalties, and the highest score of all submissions will be recorded as the final score. Students can also solve the problems in any order and can return to any of the problems at any time before the deadline for submissions.

Each question offers students the description of the graph database used in the problem, including the name and the attributes of the nodes and relationships, but students are not able to view the values stored in the database. Students are also able to see the description of the desired result and the format requirements for the result.

Students then can type in their queries in an online text editor and either save their code to finish later, or directly submit their code and be assessed by an auto-grader. If the queries have syntactic errors, the Neo4j status codes and error messages will be sent back to students to notify them about the errors; sometimes the expected correct syntax for some operators may be returned as well. If the queries can be executed successfully but have semantic errors, the students will receive messages showing the actual results and the expected results. Otherwise, students will receive full points on the problem. Instructors will assign teaching assistants to double-check submissions of students to make sure no hard-code queries were submitted (i.e., queries that use irrelevant/unnecessary conditions to match the expected results). We analyze 40,093 homework submissions written by 518 students.

4.2 Data Handling

Each submission record is assigned a numeric identification number by PraireLearn to protect students' privacy. Graduate and undergraduate research assistants are also trained to deal with research data following the research protocols; research assistants who took the course were not given access to any of the data until after it was anonymized.

It is also worth mentioning that the authors of this paper include former students in this course and the instructor of this course, and so the interpretation of the data is informed by empirical teaching experiences from the instructor and the learning perspectives from the student.

4.3 Overview of Homework Assignments

The ten problems are designed following topics to design, and the orders also follows the order of the course logic:

(1) Querying Nodes and Relationships (2 questions)
(2) Shortest Path: finding the *Shortest Path* between two nodes (1 question)
(3) Advanced Pattern Matching: finding nodes/relationships using complex patterns (1 question)
(4) `MERGE` with `ON MATCH`, `ON CREATE` Statements: `MERGE` matches existing nodes and binds them if they existed in the graph. Otherwise it creates new data and binds it (1 question)
(5) Update using the `FOREACH` function: FOREACH allow us to update elements in a path, or a list created by aggregation. (1 question)
(6) Simple Aggregation: Aggregate data using COUNT, SUM, or AVG functions (1 question)
(7) Advanced Combined Queries using `UNION`: Combine results of two queries using `UNION` (1 question)
(8) Advanced Combined Aggregation: Filtering the aggregation results, which requires the use of `WITH` command (1 question)
(9) Advanced Aggregation using `collect()`: `collect()` collects the elements of a group in a list (1 question)

4.4 The Journey of a Student Solving a Neo4j Problem

In this section, we use one submission example to demonstrate the process of how a student solves a Neo4j homework question. The question we chose was designed by the instructor to assess how students use the `WITH` clause to pipeline filtered aggregation results from one part of the query to the next [22], and we categorize this question into the concept 'Advanced Combined Aggregation'. The prompt for this question showed the requirement for completing this assignment without explicitly pointing out the recommended clauses to finish the question, motivating students to find the suitable Cypher clauses to use:

> Given a Graph Database with two kinds of nodes: Movie and Actor, find movie genres with an average rating greater than or equal to 4. When calculating the average, only movies released later than 2000 should be included. Return the genre and its average rating avg_ratings in a descending order.

We first demonstrate instructor's final solution and then we walk you through how a student arrived to it.

```
MATCH (m1:Movie)
WHERE m1.release_year > 2000
WITH m1.genre as movgen, avg(m1.ratings) as avgrat
WHERE avgrat >= 4
RETURN movgen, avgrat
ORDER by avgrat DESC
```

This solution firstly filters the movie with the restriction on the release year, then use `WITH` clause to pipeline the average rating of each movie genre, and then filter, return and order the the genre based on the average rating. To achieve this solution, this student had several submissions.

The student began by writing the following query:

```
MATCH (m1:Movie)
WITH m1.genre as avgenre, avg(m1.ratings) as avgrat
WHERE m1.genre >= 4
RETURN avgenre, avgrat
```

However, this student received the following error:

```
SyntaxError: Variable 'm1' not defined (line 4, column
```

8).
This 'Variable Name Undefined' error indicated that the student shouldn't reference variables not defined in the WITH clause, since WITH only pipelines its variables to the following queries. This is a very common mistake for students to encounter when they use the WITH clause (see Table 6).

The student tried eight submissions to get around this error, and finally realized that the error can be fixed by adding schema of 'm1' into the WITH clause:

```
MATCH (m1:Movie)
WHERE m1.release_year > 2000
WITH m1.genre, avg(m1.ratings) as avgrat
RETURN m1.genre, avgrat
```

However, the submission received another error: SyntaxError:Expression in WITH must be aliased (use AS). This error happens because the code did not specify the name for the 'm1.genre' for the result generated by WITH clause. This error is also very common (see Table 6). After fixing this issue, the student encountered another semantic mistake as the student forgot to sort the result. Finally, after another five trials the student successfully solved the question with the correct query.

This walk-through shows how a student completes a homework problem, and what some typical syntactic errors look like.

4.5 Submission Categorization

We partition student submissions into three categories: *syntactic errors*, *semantic errors*, and *correct solutions* (defined by Ahadi et al. [1]). *Syntactic errors* messages are returned by the Neo4j engine because the submitted code cannot be run; *semantic errors* occur when the submitted code can be run, but the returned result does not match the expected result. A *correct solution* is when the query executes, and the returned result matches the expected result. Table 1 shows the percentages for the results of students' submissions. In this paper, we will mainly examine *syntactic errors* for students' submissions and leave analyzing *semantic errors* to later studies.

Result	Percentage
Correct Solution	24% (9359)
Semantic Error	46% (17964)
Syntactic Error	30% (11664)

Table 1: Breakdown percentages of results in all Students' submissions

Table 2 breaks down the syntactic errors based on the status codes and shows the frequency of those errors based on students' submissions, which categorize the errors better compared to the Neo4j status code shown by the second column. Therefore, we categorize syntactic errors using the reasons given by the Neo4j engine to further analyze the distributions and variations of students' common errors.

5 Results

Table 3 shows the distributions of submissions for each concept. The order of the concepts are corresponding to the order they were presented to students. From the third column which indicates how many students have submitted a correct solution, it is clear that we could not use the complete rate to assess the difficulty rate due to the ceiling effect. However, we can see that the average attempts per student of the ninth and the tenth concepts are among the highest of all ten concepts. Those concepts require students to use advanced aggregation concept, meaning that they have to use less common aggregation keyword, or pipeline the aggregation result multiple times using WITH keyword.

Error Category	Neo4j Status Code	Number of Submissions	Percent of All Errors
Semantic Error	N/A	17964	46%
Invalid Input	SyntaxError	4438	11%
Variable Name Undefined	SyntaxError	2690	7%
Type Error	TypeError	698	2%
Expression in WITH Must Be Aliased	SyntaxError	542	1%
Invalid Use of Function Under This Context	SyntaxError	488	1%
All Sub queries in a UNION Must Have the Same Column Names	SyntaxError	438	1%
Cannot Use the Same Relationship for Multiple Patterns	SyntaxError	428	1%
Cannot Access Variables Declared Before the WITH/RETURN	SyntaxError	300	1%

Table 2: Breakdown Percentages of All Errors & Corresponding Neo4j Error Status Code Categorization

Table 4 shows the specific statistics corresponding to each concept. We calculated students' duration to finish each concept based on the time between their first submission time and final submission time. Due to constraints in the way we collected our data, we could not precisely measure how long each student spent working on each problem, only the time at which they made each submission. Based on the learning and teaching experience from the authors, for questions with a median time to finish of greater than 100 minutes, we find it extremely unlikely that students worked for this entire time; rather, we think it likely that the student was unable to solve the problem, left, and started working on it again later when they were able to receive assistance. From this table, Advanced Table Matching has the longest median time for students to finish. We can see that for the concept that needs WITH, the median time for students to finish is the longest, and the correctness rate is the lowest among other concepts.

Table 5 shows the breakdown percentages of all syntactic errors. Even though errors such as 'Variable Name Undefined' has the comparatively higher percentages in occurred syntactic errors, students could have multiple potential syntactic mistakes to receive this error, and under most of the situations the error should be classified further into errors such as 'Cannot Access Variables Declared Before the WITH/RETURN'. In Section 6, we will further discuss about how error classification could affect students.

Concept	# Submissions	# Students Who Attempted	# Students who completed	Average Submissions per Student
Simple Querying Nodes/Relationships	3951	518	507	7.79
Advanced Querying Nodes/Relationships	4259	518	507	8.4
Shortest Path	3401	517	510	6.67
Advanced Pattern Matching	3715	516	500	7.43
Graph Update (I) `ON MERGE, ON CREATE`	2338	516	514	4.55
Graph Update (II), `FOREACH`	2683	517	511	5.25
Simple Aggregation	2702	512	506	5.34
Advanced Combined Queries, `UNION`	3984	512	509	7.82
Advanced Combined Aggregation	6735	511	497	**13.55**
Advanced Aggregation, `collect()`	5277	511	505	**10.44**

Table 3: Number of Submissions per Question

Concept	Median Time to Finish (Hours:Minutes)	Correct	Syntactically Wrong	Semantically Wrong
Simple Querying Nodes and Relationships	0:51	22%	20%	57%
Advanced Querying Nodes and Relationships	1:09	23%	25%	53%
Shortest Path	1:35	26%	29%	45%
Advanced Pattern Matching	**3:56**	26%	28%	46%
Graph Update with `ON MERGE, ON CREATE`	0:26	43%	31%	26%
Graph Update with `FOREACH`	0:28	34%	46%	20%
Simple Aggregation	0:37	33%	36%	31%
Advanced Combined Queries, `UNION`	0:40	23%	45%	32%
Advanced Combined Aggregation	**3:07**	14%	25%	61%
Advanced Aggregation using `collect()`	2:07	19%	26%	55%

Table 4: Breakdown of errors by Neo4j concept evaluated

Table 6 shows the distributions of errors for each concept. The eighth column has a drastic increase in the concept of `Advanced Pattern Matching`, which echoes with the finding in Table 4, indicating that students having difficulty in organizing queries with multiple relationships and nodes. Another intriguing fact is the spiking number of error occurrences after the 'Simple Aggregation' for the 'Variables Name Undefined' error (the third column), which indicates that only in questions that need `WITH` clauses, this category jumps to the most frequent error that students tend to have. This error also persists over multiple problems, implying that students have problems understanding this error message properly when they use `WITH` clauses.

6 Discussion

As first noted in our data exploration and confirmed by the results in Table 5, students commonly encounter the error `'Variable Name Undefined'` when they use `WITH` because they forget to reference all variables that needs to be used in the `RETURN` or in `WHERE` clause. From our learning and teaching experiences, we suspect that this is because of the less-intuitive design of how to filter aggregation results in Neo4j: students have to use `WITH` to select aggregation results to filter it later in the `WHERE` or `RETURN` clause, but from what we observe in students' submissions, we find out students tend to directly use `WHERE` clause to filter the aggregation results directly, and they will receive this `'Variable Name Undefined'` error. As shown in the table, students do not only make this error on one problem type, but continue to encounter it on subsequent questions, despite having successfully solved multiple problems that required a `WITH` statement.

Furthermore, when students correctly use `WITH` to aggregate variables, but forget to define variables in the `WITH` clause and reference them in latter clauses, there are two kinds of error messages. The `Variable Name Undefined` error happens when students reference them in `RETURN` clauses. The other error category, listed in the last column of Table 6 only appears if students reference those variables in `WHERE` clauses. Even though this error message pointing out where students make mistakes, it instead makes students ignore that they would make the same mistakes in `RETURN` clauses. Thus, we suggest it would be better if the error messages can classify the situation more clearly. Cypher, as McCall and Kölling mentioned, shares the same conceptual mistake as Java that can manifest itself in different error messages, making it very challenging for novice programmers to make progress [19].

7 Limitations & Future Work

In this paper, we only utilize the homework submissions to speculate students' learning behaviors without test submissions due to the lack of sufficient data. Because we cannot ensure that students' techniques used in finishing their homework assignments, the reliability of the data needs to be further validated. The data source is only coming from one university and one course, making the data less universal and may be limited to the design deficiency of the courses and the programming levels of the university students. There are only one to two questions designed for each concept, making it harder to compare and speculate the difficulty rates for

Error (Neo4j)	% of all Syntactic Errors
Invalid Input	38%
Variable Name Undefined	23%
Type Error	6%
Expression in `WITH` Must be Aliased	5%
Invalid Use of Function Under This Context	4%
All Sub Queries In an `UNION` Must Have the Same Column Names	4%
Cannot Use the Same Relationship for Multiple Patterns	4%
Cannot Access Variables Declared Before the `WITH/RETURN`	3%
`RETURN` Not Used Correctly	2%
Unexpected End of Input	2%
Unknown Function	2%

Table 5: Syntactic Error Percentages

Concept (Neo4j)	Invalid Input	Variable Name Undefined	Type Error	Expression in `WITH` Must Be Aliased	Invalid Use of Function Under This Context	All Sub Queries in an `UNION` Must Have the Same Column Names	Cannot Use Same Relationship for Multiple Patterns	Cannot Access Variables Declared Before the `WITH` / `RETURN`
Simple Querying Nodes and Relationships	503	120	18	31	4	0	3	17
Advanced Querying Nodes and Relationships	620	152	7	10	2	12	48	1
Shortest Path	549	68	53	3	1	0	0	4
Advanced Pattern Matching	339	194	31	38	6	5	**236**	18
Graph Update with `ON MERGE`, `ON CREATE`	456	89	71	0	0	2	0	0
Graph Update with `FOREACH`	531	196	305	6	32	1	0	7
Simple Aggregation	235	**287**	64	**235**	40	3	0	**69**
Advanced Combined Queries, `UNION`	297	**531**	42	**67**	180	407	3	**34**
Advanced Combined Aggregation	474	**591**	52	**78**	113	6	83	**72**
Advanced Aggregation using `collect()`	432	**464**	59	**74**	111	0	55	**81**

Table 6: Error submissions per Concept

each concept in Neo4j, as the results might be greatly affected by the wording of the questions. In the future, we can further design the course content to serve the purpose of comparison between students' performances in different classes.

Another limitation is the ceiling effects discussed in the previous section. Because the overall percentage of correctness among students is high, failed numbers may be affected by random factors based on individual's learning preferences. We also only analyze based on error messages without investigating the students' submissions, which isn't sufficient for us to understand the semantic errors. More qualitative analysis of submissions could give greater insight into student's semantic errors.

For future work, it would also be very intriguing if we could partition the students based on their final grades in this course and study their learning behaviors separately in comparison, which may lead to further discussions on the correlation between how students learn through their mistakes and how they perform on the overall course content, providing educators with more insights. We could also use a qualitative approach to interview several students in the course, asking them to talk aloud through their thought processes while solving the homework problems. This would give great insights into why students make the mistakes that they do.

8 Conclusion

There has been almost no research up to this point on graph-database education. In this paper, we build on our previous work on SQL database learning to understand how students learn graph databases. We use over 40 thousand students' submissions to analyze syntactic errors students encounter while solving Neo4j homework problems. Our analysis shows that students often encounter difficulties in understanding the syntax of `WITH`, even after working through multiple questions requiring this construct.

Acknowledgments

This work supported in part by NSF IUSE grant 2021499.

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 (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 (ITiCSE '15)*. ACM, New York, NY, USA, 201–206. https://doi.org/10.1145/2729094.2742620

[3] Abdussalam Alawini. 2018. *Database Systems*. https://alawini.com/teaching/cs411-database-systems/

[4] Ridha Alkhabaz, Seth Poulsen, Mei Chen, and Abdussalam Alawini. 2021. Insights from Student Solutions to MongoDB Homework Problems. In *Proceedings of the 2021 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '21)*. Association for Computing Machinery, New York, NY, USA, 7.

[5] Peter Brusilovsky, Sergey Sosnovsky, Michael V. Yudelson, Danielle H. Lee, Vladimir Zadorozhny, and Xin Zhou. 2010. Learning SQL Programming with Interactive Tools: From Integration to Personalization. *ACM Trans. Comput. Educ.* 9, 4, Article 19 (Jan. 2010), 15 pages. https://doi.org/10.1145/1656255.1656257

[6] Mike Buerli. 2012. The current state of graph databases. *Department of Computer Science, Cal Poly San Luis Obispo, mbuerli@ calpoly. edu* 32, 3 (2012), 67–83.

[7] Edgar F Codd. 2002. A relational model of data for large shared data banks. In *Software pioneers*. Springer, 263–294.

[8] Susan Davidson. 2020. *Data Management in the Cloud*. https://www.seas.upenn.edu/~cis550/

[9] María del Pilar Ángeles. [n.d.]. *NoSQL systems*. https://www.coursera.org/learn/nosql-databases

[10] Diogo Fernandes and Jorge Bernardino. 2018. Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB.. In *DATA*. 373–380.

[11] Ghislain Fourny. 2019. *Big Data*. https://video.ethz.ch/lectures/d-infk/2019/autumn/263-3010-00L.html

[12] Brad Fowler, Joy Godin, and Margaret E Geddy. 2016. Teaching Case: Introduction to NoSQL in a Traditional Database Course. *J. Inf. Syst. Educ.* 27 (2016), 99–104.

[13] José Guia, Valéria Gonçalves Soares, and Jorge Bernardino. 2017. Graph Databases: Neo4j Analysis. In *ICEIS*.

[14] José Guia, Valéria Gonçalves Soares, and Jorge Bernardino. 2017. Graph Databases: Neo4j Analysis.. In *ICEIS (1)*. 351–356.

[15] Amarnath Gupta. [n.d.]. *Graph Analytics for Big Data*. https://www.coursera.org/learn/big-data-graph-analytics

[16] Dimitrios Kotsifakos, Dimitrios Magetos, Alexandros Veletsos, and Christos Douligeris. 2019. Teaching the Basic Commands of NoSQL Databases Using Neo4j in Vocational Education and Training (VET). *European Journal of Engineering Research and Science* CIE (Apr. 2019), 13–18. https://doi.org/10.24018/ejers.2019.0.CIE.1291

[17] Josep Lluís Larriba-Pey, Norbert Martínez-Bazán, and David Domínguez-Sal. 2014. *Introduction to Graph Databases*. Springer International Publishing, Cham, 171–194. https://doi.org/10.1007/978-3-319-10587-1_4

[18] João Ricardo Lourenço, Bruno Cabral, Paulo Carreiro, Marco Vieira, and Jorge Bernardino. 2015. Choosing the right NoSQL database for the job: a quality attribute evaluation. *Journal of Big Data* 2, 1 (2015), 1–26.

[19] Davin McCall and Michael Kölling. 2014. Meaningful categorisation of novice programmer errors. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*. IEEE, 1–8.

[20] Justin J Miller. 2013. Graph database applications and concepts with Neo4j. In *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*, Vol. 2324.

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

[22] Neo4j, Inc. 2019. Neo4j. https://neo4j.com/

[23] 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 (ITiCSE '20)*. Association for Computing Machinery, New York, NY, USA, 404–410. https://doi.org/10.1145/3341525.3387391

[24] Rabi Prasad, Padhy Manas, Ranjan Patra, and Suresh Chandra Satapathy. 2011. RDBMS to NoSQL: Reviewing Some Next-Generation Non-Relational Databases. , 15-30 pages.

[25] Phyllis Reisner. 1981. Human Factors Studies of Database Query Languages: A Survey and Assessment. *ACM Comput. Surv.* 13, 1 (March 1981), 13–31. https://doi.org/10.1145/356835.356837

[26] Facts & Factors Research. Feb, 2020. *Graph Database Market By Product Type (Resource Description Framework and Property Graph), and By Application (BFSI, IT & Telecom, Healthcare & Life Sciences, Transportation & Logistics, Retail & Ecommerce, Government & Public, and Others): Global Industry Perspective, Comprehensive Analysis, and Forecast, 2019 - 2026*. https://www.fnfresearch.com/graph-database-market-by-product-type-resource-description

[27] Kristin Tuft and David Maier. 2014. *Data Management in the Cloud*. http://datalab.cs.pdx.edu/education/clouddbms-win2014/page.php?content=index

[28] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins. 2010. A Comparison of a Graph Database and a Relational Database: A Data Provenance Perspective. In *Proceedings of the 48th Annual Southeast Regional Conference (ACM SE '10)*. Association for Computing Machinery, New York, NY, USA, Article 42, 6 pages. https://doi.org/10.1145/1900008.1900067

[29] 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.