A Quantitative Analysis of Student Solutions to Graph Database Problems

Anonymous Author(s)

Abstract

As data grow both in size and in connectivity, there has been a growing interest in using graph databases in industry. 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 their submissions writing different types of queries in Cypher, the query language for Neo4j–the most prominent graph database. Based on 40,093 student submission from homework assignments in an upper-level computer science database course, this paper provides insights and quantitative analysis about students' learning when solving graph database problems. The data show that writing more complex queries initially takes students more time and more attempts to get right. Additionally, students struggle to correctly use Cypher's `WITH` clause to define variable names, and these errors persist over multiple homework problems requiring the same techniques.

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:

Anonymous Author(s). 2020. A Quantitative Analysis of Student Solutions to Graph Database Problems. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

In many fields data is represented in graph form, which is one of the fundamental data abstractions in the field of computer science that emphasizes not only the data itself but also the relationships between data [8]. Relational database have been used for decades [11], 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 [21]. 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.

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-6874-2/20/06…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In recent years, the popularity of graph databases has spiked drastically: in 2017, over half of the enterprise users across all industry utilize graph database due to its speed and enhanced execution. The report also points out global graph database market in 2019 is approximately 810 Million USD and is anticipated to be 4.5 billions USD in the next decade [1].

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. It is one of the most popular graph database: eBay use Neo4j in its eBay App for Google Assistant in order to support the probabilistic models and aid understandings in the conversational shopping scenario; NASA use Neo4j in its knowledge architecture 'Lesson Learned Database' to boost the efficiency of extracting knowledge from connected areas; other industries, Walmart, Cisco, Airbnb etc. [25] 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 [14], Neo4j stands out among the current graph database thanks to its simplicity, agility and flexibility. Vicknair et al. [28] also used Neo4j to show that graph system shows better performance in full-text character searches and structural type queries compared to that in relational database.

The need for introducing students to graph databases is also rising with potential pervasive usage in chemistry, biology, semantic web, social networking and recommendation engines [10, 23]. Multiple universities have opened courses that incorporate graph database into their topics: Portland State University offered Neo4J in its 'Data Management in the Cloud' course as early as 2014 [20]; the University of Pennsylvania's Database and Information Systems course has taught Neo4J since 2017 [12]; our university's 'Database Systems' course, taught by the last author, also covers Neo4j since 2018 [6]. Additionally, Coursera, the online learning platform, offers introductory Neo4j courses [13, 18].

In response to the need of graph database talents and the lack of CS education research on how students learn graph database, we study the following 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?
(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 the rise of data that are bigger both in size and in interconnectivity, there is a trend to utilizing non-relational database in the area of grid and cloud computing [22, 26], as they are more efficient, distributed,

schema-free compared to the traditional database management system.

However, Computer Science Education research on NoSQL database languages is significantly lacking compared to that on SQL databases. Ahadi et al. [2, 3] provides a quantitative analysis about relative difficulties in SQL queries. The author's previous work [7] uses quantitative approaches to identify the most common students' errors when writing SQL queries; other papers study the potential difficulties students face from ease-of-use perspective [27]; other researchers proposed novel tutors and tools to help with teaching SQL [9]. There are rare studies conducted about NoSQL database. Fowler [15] demonstrated a successful teaching case that incorporated NoSQL database into the traditional database management course to students and received significant improvement in understandings NoSQL. Sriram [24] also proposed a four-tiered learning model to better teach NoSQL Databases to undergraduate students.

We have only found one study about graph database in Computer Science Education [19] that conducted a Neo4j teaching case that shows the successful improvements in Vocational Education and Training environment. Other research on graph database is mainly about performance analysis on different operating systems or introductions on the differences between graph database and RDBMS or other NoSQL database [16, 28].

To the best of our knowledge, there has been no research in understanding the mistakes students make while learning to 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 [14, 17].

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.

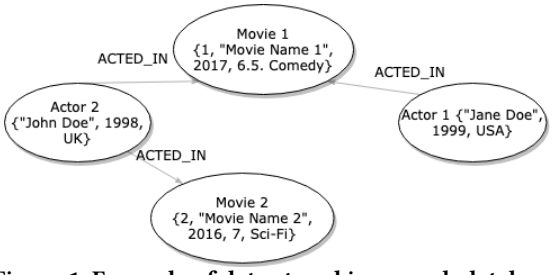

Figure 1: Example of data stored in a graph database.

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)
> **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.

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

Here we can see clearly the relationship of actors and movies in the Neo4j, and compared to the SQL query, Neo4j 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 (Sections 4.1, 4.2 and 4.3), and then we present an example of a student solving a homework problem. In Section 4.5, we discuss how we categorized students' submissions.

4.1 Data Collection

The data was collected by parsing homework submissions made by students taking an upper-lever Computer Science course in a large public U.S. university. Students submit their assignments on an online homework and exam platform, which we will call OHEP for anonymization purposes [5]. The students are allowed 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 of submissions.

There are 10 problems in the Neo4j homework. 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. In the latter case, their submitted queries will be assessed by an auto-grader that is connected to an online Neo4J database. 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. The Neo4j engine may also return error messages with the expected correct syntax for some operators. 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 data from student submissions from the Fall 2019 and Spring 2020 offerings of a database course, which is divided to four modules:

- *data models and query languages*
- *relational database design*
- *relational database system internals*
- *advanced database topics*

In the *data models and query languages* module, students learn about SQL and MongoDB, and then Neo4j. 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 classes. We analyze 40,093 submissions written by 518 students.

4.2 Data Handling

Each submission records is assigned with a numeric identification numbers by OHEP in order to protect students' privacy. Graduate and undergraduate research assistants are also trained to deal with research data following the research protocols; research assistance 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 a former student 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:

(1) Querying Nodes and Relationships (2 questions)
(2) Shortest Path (1 question)
(3) Advanced Pattern Matching (1 question)
(4) `MERGE` with `ON MATCH`, `ON CREATE` Statements (1 question)
(5) Update with `FOREACH` (1 question)
(6) Simple Aggregation (1 question)
(7) Advanced Aggregation using `collect()` (1 question)
(8) Advanced Combined Queries(1 question)
(9) Advanced Combined Queries, `UNION` (1 question)

The order of the questions also follows the order of the course logic in class.

4.4 The Journey of a Student Solving a Neo4J Problem

We here demonstrate how a student solve 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 [25]. 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 of avg_ratings.

The student began by writing the following query:

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

The student's first submission has the following error:
`SyntaxError: Variable 'm1' not defined (line 4, column 8)`.
The error message indicates that the student mistakenly reference the variable '`m1`' in the `RETURN` clause without mention it in the `WITH` clause. 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
WHERE avgrat >= 4
RETURN m1.genre, avgrat
```

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

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

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 following query:

```
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 realistic walk-through shows how a typical student might go through a homework problem.

4.5 Submissions Categorization

We partition student submissions into three categories: *syntactic errors*, *semantic errors*, and *correct solutions* (defined by Ahadi et al. [2]). *Syntactic errors* messages are returned by the Neo4j engine because the submitted code cannot be run; *semantic errors* occurred 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.

Table 2 breaks down the syntactic errors based on the status codes and show the frequency of those errors based on students' submissions. 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 & Discussion

One way to show which types of questions are more difficult for students would be to look at the percentage of students who successfully completed each problem. But this is infeasible for us because almost every student in our data set successfully completed every problem (see Table 3), casusing major ceiling effects. Instead, we use the number of attempts that students took to solve each problem, as well as the median time between the student's first and last submission (see Table 4), to try to understand which types of questions were more difficult. In Table 3, the distribution of submissions is about even across every question except the questions about "Advanced Combined Aggregation" that have higher submissions. Those concepts require students to use `WITH` or `UNION` keyword to combine two sub-queries together.

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%
`RETURN` Not Used Correctly	SyntaxError	283	1%
Unexpected End of Input	SyntaxError	239	1%

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

Table 4 shows the specific statistics corresponding to each concept. The order of the concepts are corresponding to the order they were presented to students. 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 have no way of knowing how long each student actually 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. Also, from this table we can clearly see that on most concepts, comparatively longer median time students spent to finish each question, the lower the correctness they have and the lower overall distribution of syntactic errors for the overall corresponding submissions.

Table 5 shows the breakdown percentages of all syntactic errors. Aside from generic errors (e.g. Invalid input, Undefined variables, and Type errors), the most common syntactic errors were Neo4j-specific (e.g. usage of `WITH` and `UNION`), indicating that students have difficulty with Neo4j-specific concepts.

Table 6 shows the distributions for each errors for each concept. One intriguing fact is the spiking after the `'Simple Aggregation'` for the undefined error (the second biggest error). In Section 4.4, we once proposed that the 'Variable name undefined' errors often

Concept	# Submissions	# Attempted Questions	# Completed Questions
Simple Querying Nodes and Relationships	3951	518	507
Advanced Querying Nodes and Relationships	4259	518	507
Shortest Path	3401	516	509
Advanced Pattern Matching	3715	516	500
Graph Update with `ON MERGE, ON CREATE`	2338	516	514
Graph Update with `FOREACH`	2683	517	511
Simple Aggregation	2702	512	506
Advanced Combined Queries, `UNION`	**5226**	512	506
Advanced Aggregation	**6735**	511	497
Advanced Aggregation using `collect()`	3977	510	507

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%
Complex 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`	**4:31**	19%	35%	46%
Advanced Aggregation	3:07	14%	25%	61%
Advanced Aggregation using `collect()`	0:30	24%	33%	44%

Table 4: Breakdown of errors by Neo4j concept evaluated

occurs when students first start to use `WITH` and forget to reference all variables that needs to be used in the `RETURN` or in `WHERE` clause. We believe that the spikes actually represent this common error, because only in the last four concepts this error jumps to be the biggest error, and all those questions are usually completed via the usage of `WITH`. 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 in order to filter it later in the `WHERE` 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.

The climax of the number of occurrences in Table 6 we can see its spike in the first concept that requires the use of `'WITH'` and only barely afterwards even though all the following concepts require the advanced usage of `'WITH'`. As mentioned in section 4.4 as well, this error occurs when students forget to give the newly manipulated result a new name (schema). This is a straight-forward error that students tend to forget when they first use `WITH` clause; but once they have encountered such error, they learn to rename it in later queries.

6 Limitations & Future Work

In this paper we only utilize the homework submissions to speculate students' learning behaviors instead of using test submissions due to the lack of sufficient data. Because we cannot ensure that students use what kind of techniques to finish their homework assignments, the reliability of the data needs to be further validated. The data source is also only coming from one university and one course, which makes 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, makes it harder to compare and speculate the difficulty rates for each concepts 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 are so high, failed numbers may be affected be random factors such as individual's learning preferences. We also only analyze based on error messages without investigating the students' submissions, which makes us unable to sufficiently understand the semantic errors. More qualitative analysis of submissions, as done by Ahadi et al. for SQL [4], 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 with 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.

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`	`RETURN` not Used Correctly
Simple Querying Nodes and Relationships	503	120	18	31	4	0	3	17	36
Advanced Querying Nodes and Relationships	620	152	7	10	2	12	48	1	24
Shortest Path	549	68	53	3	1	0	0	4	11
Advanced Pattern Matching	339	194	31	38	6	5	236	18	17
Graph Update with `ON MERGE`, `ON CREATE`	456	89	71	0	0	2	0	0	14
Graph Update with `FOREACH`	531	196	305	6	32	1	0	7	6
Simple Aggregation	235	**287**	64	**235**	40	3	0	69	15
Advanced Combined Queries, `UNION`	424	**592**	49	**101**	226	189	25	71	66
Advanced Aggregation	474	**591**	52	**78**	113	6	83	72	20
Advanced Aggregation using `collect()`	307	**401**	48	**40**	64	220	33	41	74

Table 6: Error submissions per Concept

7 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 student learn graph databases. We use over 40 thousands students' submissions to analyze syntactic errors students encounter while solving Neo4J homework problems. Our analysis shows that students often spends longer time trying to finish questions that involve complex Cypher constructs that they are not familiar with. Students often encounter difficulties in understanding the syntax of `WITH`, even after working through multiple questions requiring this construct.

References

[1] 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

[2] 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

[3] 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

[4] Alireza Ahadi, Julia Prior, Vahid Behbood, and Raymond Lister. 2016. Students' Semantic Mistakes in Writing Seven Different Types of SQL Queries. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16)*. ACM, New York, NY, USA, 272–277. https://doi.org/10.1145/2899415.2899464

[5] Anonymous. 2015. Paper Describing an Online Learning System.

[6] Anonymous. 2018. *Database Systems*. http://link.to.course.webpage.com/

[7] Anonymous. 20XX. Paper analyzing student's SQL homework solutions.

[8] Claude. Berge. 1962. *The theory of graphs and its applications / by Claude Berge ; translated by Alison Doig*. Methuen ; Wiley London : New York. x, 247 p. : pages.

[9] 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

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

[11] E. F. Codd. 1970. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM* 13, 6 (June 1970), 377–387. https://doi.org/10.1145/362384.362685

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

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

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

[15] 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.

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

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

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

[19] 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

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

[21] 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

[22] João Ricardo Lourenço, Bruno Cabral, Paulo Carreiro, Marco Vieira, and Jorge Bernardino. [n.d.]. Choosing the right NoSQL database for the job: a quality attribute evaluation. 2, 1 ([n. d.]), 18. https://doi.org/10.1186/s40537-015-0025-0

[23] 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.

[24] 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

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

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

[27] 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

[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