Insights from Student Solutions to SQL Homework Problems

Seth Poulsen
sethp3@illinois.edu
University of Illinois at Urbana-Champaign

Liia Butler
liiamb2@illinois.edu
University of Illinois at Urbana-Champaign

Abdussalam Alawini
alawini@illinois.edu
University of Illinois at Urbana-Champaign

Geoffrey L. Herman
glherman@illinois.edu
University of Illinois at Urbana-Champaign

Abstract

We analyze the submissions of 286 students as they solved Structured Query Language (SQL) homework assignments for an upper-level databases course. Databases and the ability to query them are becoming increasingly essential for not only computer scientists but also business professionals, scientists, and anyone who needs to make data-driven decisions. Despite the increasing importance of SQL and databases, little research has documented student difficulties in learning SQL. We replicate and extend prior studies of students' difficulties with learning SQL. Students worked on and submitted their homework through an online learning management system with support for autograding of code. Students received immediate feedback on the correctness of their solutions and had approximately a week to finish writing eight to ten queries. We categorized student submissions by the type of error, or lack thereof, that students made, and whether the student was eventually able to construct a correct query. Like prior work, we find that the majority of student mistakes are syntax errors. In contrast with the conclusions of prior work, we find that some students are never able to resolve these syntax errors to create valid queries. Additionally, we find that students struggle the most when they need to write SQL queries related to `GROUP BY` and correlated subqueries. We suggest implications for instruction and future research.

CCS Concepts

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

Keywords

SQL, database education, online assessment

ACM Reference Format:
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), June 15–19, 2020, Trondheim, Norway.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3341525.3387391

ITiCSE '20, June 15–19, 2020, Trondheim, Norway
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6874-2/20/06…$15.00
https://doi.org/10.1145/3341525.3387391

1 Introduction

Databases are pervasive and vital to the use, and security of, sensitive data such as in medical, financial, scientific, and consumer contexts [7]. They are used not only by software developers, but by data scientists, researchers, actuaries, accountants, business analysts, product managers, and people working in finance and marketing. People with data-related skill are becoming more and more in demand [25]. Structured Query Language (SQL) is the de facto standard for querying relational databases [17]. Indeed, by some metrics, it is the most in demand programming language [17, 18]. Knowledge of SQL is vital to the use of data science for accelerating scientific inquiry, business decision-making, and effective healthcare. Critically, SQL does not require prerequisite knowledge of other programming language paradigms and can be immediately useful to a wide range of professionals, providing an otherwise overlooked and underused gateway into computing disciplines.

Writing queries in SQL could very well be a students' first programming experience and their first introduction to formal computing. Unlike the wealth of research on students' difficulties in learning imperative, object-oriented, or functional languages, there has been little research on how students learn SQL. SQL is a declarative query language, which is a completely different paradigm than other common introductory languages such as Python, Java, or C. Consequently, the majority of misconceptions and programming difficulties research conducted in these paradigms does not translate well, if at all, to how people learn SQL. In this paper, we contribute to the nascent research literature on how students learn SQL by documenting the types of mistakes that students make when first learning to write database queries using SQL and their ability to surmount those mistakes.

In response to recent calls to increase replication studies in Computing Education Research [10], we replicate and extend recent studies by Ahadi et al. and Taipalus and Perälä [1, 2, 21]. Replication studies in education research are vital for determining the replicability of prior findings but also whether findings generalize to other contexts and other students who have experienced different curricula and modes of instruction. Ahadi et al. classified student errors when writing SQL queries from summative exams, while Taipalus and Perälä looked at homework submissions. We classify 52093 SQL queries written by 286 students over the Summer and Fall semesters of 2019 while working on formative homework problems for an upper-level databases course at the University of Illinois at Urbana-Champaign, a large, public university in the Midwestern United States. This analysis contributes to the small but growing body of knowledge on what mistakes students may typically make while learning to write databases queries.

2 Literature Review

There has been far less research conducted on teaching SQL than for teaching students to program in imperative or functional languages. A few groups of educators have created and assessed intelligent tutors to help teach SQL [6, 13]. Other researchers have tried to understand the common mistakes and pitfalls that people run into while trying to learn to write queries in SQL. Reisner took a human-factors approach, evaluating SQL as a user interface and finding that students had a particulary hard time constructing queries requiring correlated subqueries or GROUP BY clauses. [19, 20]. Welty and Stemple followed Reisner's lead in methodology, verifying her findings and gaining additional insight to the similar and different struggles of SQL when compared to TABLET, which was a competing query language at the time [23]. Ahadi et al. took a more quantitative approach, analyzing about 160,000 SQL queries written by about 2,300 students to try and understand the most common errors that students make [1–3]. They also did some qualitative analysis of the student queries to try to understand why certain errors were so common [3]. Most studies so far have focused on the struggles of students during summative assessments, leaving a gap in the literature about the struggles students experience while working on SQL homework assignments, which has only just begun being filled by the work of Taipalus and Perälä [21].

All of the above studies agree that students struggle more to complete queries that include a subquery or a join than queries that operate only on a single table, without any joins or subqueries. Ahadi et al. [1] and Welty and Stemple [23] both also point out GROUP BY clauses as being particularly difficult for students to construct. By contrast, Reisner [19] found that the primary difficulty with GROUP BY was students not being able to figure out when they should use it or not, but that they were able to construct a query using it properly once they knew if it was needed. Both Reisner [20] and Ahadi et al. [3] note that not having adequate mental strategies for solving the problems was a significant barrier to students success constructing SQL queries. Reisner, as well as Welty and Stemple, studied students who had little to no prior programming experience, whereas Ahadi and his coauthors studied students who already had a few years of programming experience. The students in all of the studies had little to no experience using a database query language.

Here we present a replication of the results of prior research in understanding student behaviours while learning SQL, and extend them by including data from homework problems dealing with left outer joins, triggers, stored procedures, and functions, features of SQL that have not been included in any prior study on learning of SQL.

3 Data Collection

The University of Illinois at Urbana-Champaign is a research-intensive, primarily-white institution with 2000 undergraduate CS majors. Databases is an elective course taken primarily by upper-level undergraduate and graduate students with prerequisites including introduction to programming and data structures. The course teaches students about database systems covering three main modules: *data models and query languages* (relational model: relational algebra and SQL, graph model: Neo4J, and document-oriented model: MongoDB), *database design* (conceptual design and normal forms), and *database implementation* (storage and indexing, query execution and optimization, concurrency control) with only five class meetings devoted to teaching students about SQL queries. We collected data from the Summer and Fall 2019 offerings of the course, which includes 8 and 10 assigned questions about SQL queries, respectively. Our sample includes submissions from 286 students, including 95 female and 191 male students.

3.1 Description of Homework Assignments

Students' assignments were delivered and graded by the online learning management system PrairieLearn [24]. PrairieLearn provides support for the random generation of homework problems and the autograding of students' code submissions. The students are allowed unlimited attempts to solve each homework problem before the due date and receive feedback on the correctness of their submission after each attempt. Students can attempt problems in any order and can leave and return to a problem without penalty.

For SQL questions, students are given the database schema relevant to the problem (i.e., they can see the table names and the name and type of each data value in the tables). Students can not see the items in the database itself. Below the schema, the students see the question prompt that their query must address. The database management system used for the queries is MySQL [16]. The students have a simple code editor to write their query. The students have the option to save their submission, or select to save and grade.

When the student selects to have their submission graded, their submitted query is run against the database. Should the query have a syntactic error, the MySQL error code and message is reported back and displayed to the user. If the query successfully runs, the student can see whether the query was correct (received full points) or contained a semantic error (did not receive full points). The expected results and actual results are displayed for feedback. Students are graded on functional correctness (i.e., the actual results matched the expected results). Efficiency or use of intended SQL features does not factor into score, the exception being that teaching assistants screen submissions after the fact to ensure that all queries are actually pulling the data from the database, not just writing a query that hardcodes what they know the expected answer is. To avoid hardcoding, the autograder also generates random data instance every time a student submits a query.

3.2 Data Handling

To protect students' data, PrairieLearn assigns each student a random numeric identifier so that an individual student's data can be tracked without revealing personally identifiable information. All graduate research assistants handling the data were trained in human subjects research protocols.

3.3 A First Look at the Data

We present a few submissions from a series of submissions from one student solving a problem. The example comes from a homework question that the instructor intended students to solve by using a JOIN and a WHERE clause. The prompt for this question showed the schema for each table in the database (omitted due to space limitations), along with the following text:

> This schema has tables for the customers of the mobile phone store, purchases made there, products sold there, and brands of the products sold. The primary key for each table is underlined.
>
> The `Customers` table includes a unique `CustomerId` for each customer. The `Purchases` table contains a unique `PurchaseId` for each transaction. For the sake of simplicity, we will assume that a customer can only purchase one item at a time. The `Products` table includes a unique `ProductId` for each product and other relevant information. The `Brands` table includes a unique `BrandName` for each company.
>
> Using this mobile phone store schema, write a SQL query that returns the name of each brand and the number of brands that were established before it.

It is important to note that though the instructor intended the student to solve this problem using a `JOIN`, the instructor did not explicitly tell the students which SQL features to use, following the philosophy that students will learn how to use SQL better if they have to figure out which features to use on their own. In fact, some students solved this problem using a subquery rather than doing a self-join. The student whose work we examine started with the following submission, in which it appears that they were attempting to make the comparison between rows using a subquery:

```
SELECT A.BrandName, Count(*)
FROM Brands A, (
    SELECT *
    FROM Brands B
    WHERE A.YearEstablished < B.
        YearEstablished
)
```

This submission generated the following error:

```
1248 (42000) Every derived table must have its own alias
```

To resolve that error, the student added an alias for the table generated by their subquery, but then received another error:

```
1054 (42S22) Unknown column 'A.YearEstablished'
   in 'where clause'
```

After a few more small changes to their query, each of which yielded another error, the student backtracked and wrote an entirely different query. It appears here that they were not trying to write a correct answer, but perhaps to just get something that worked, and explore the data set a little bit:

```
SELECT "Apple", Count(*)
FROM Brands
WHERE YearEstablished < "1976"
```

They then generalized this new query to one with an implicit self join:

```
SELECT BrandName, Count(*)
FROM Brands B, Brands C
WHERE C.YearEstablished < B.YearEstablished
GROUP BY B.BrandName
```

but this gave yet another error:

```
1052 (23000) Column 'BrandName' in field list is
   ambiguous
```

With some more exploration and tweaks, the student was eventually able to produce a correct solution to the problem:

```
SELECT B.BrandName, Count(*)
FROM Brands B, Brands C
WHERE C.YearEstablished <= B.YearEstablished
GROUP BY B.BrandName
```

This brief run-through of a student's work gives an idea, at a high level, of the type of information we have available in our data set. From this example, we can see that this student struggled through several syntactic errors for each strategy they attempted. The student had many variants for a single approach and built upon what they had incrementally.

4 Methodology

We follow the basic methodology of Ahadi et al. [1, 2] by dividing student mistakes into *syntactic errors*–errors where the SQL engine was not able to execute the query and thus returned an error code, and *semantic errors*–errors where the SQL engine was able to run the query and obtain a result set, but the result set was incorrect.

In this paper, we focus mostly on further categorization of syntactic errors, leaving analysis of semantic errors to later work (those interested in categorization of semantic errors should see [3, 21, 22]). We categorize syntactic errors using the error codes generated by the MySQL query engine [16].

We use this analysis to answer the following research questions:

- What are the distributions of submissions resulting in syntactic errors, semantic errors, and correct submissions in solving SQL homework problems and how do they vary between question concepts?
- What types of syntactic errors are made by students who are successful or unsuccessful when solving SQL homework problems?

5 Results

Table 1 shows the breakdown of syntactic vs. semantic errors by which concept a given exercise was evaluating, showing that students struggle more to complete more complex queries. The total number of submissions for each group of questions can be seen in table 2, along with the number of attempts and successes for questions in that category. This gives some insight into the average number of attempts a student took before completing each problem.

The breakdown of errors by error code in table 3 shows that almost 40% of the time a student receives an error message from the MySQL engine, it is a syntax error. Other errors, such as undefined

Concept	Correct	Syntactic Error (eventually correct)	Semantic Error (eventually correct)
Join and Group By	8%	53% (100%)	36% (99%)
Join and Where	14%	39% (99%)	43% (99%)
Join, Where, and Distinct	27%	36% (100%)	32% (100%)
Table update, Simple Query	19%	45% (99%)	31% (99%)
Left Outer Join and Group By	7%	46% (98%)	44% (98%)
Delete with Subquery	10%	64% (94%)	22% (95%)
Correlated Subquery and Group By	7%	44% (95%)	45% (95%)
Triggers	13%	71% (95%)	11% (97%)
Stored Procedures and Functions	6%	67% (96%)	21% (99%)

Table 1: Breakdown of errors by SQL concept evaluated. This gives some surface-level insight into which types of homework problems are more difficult. Students are less likely to complete exercises requiring the use of more advanced SQL features.

Concept (# Questions)	# Submissions	# Student Attempts	# Student Successes
Join and Group By (2)	6570	283	282
Join and Where (2)	4354	280	278
Join, Where, and Distinct (2)	1812	278	278
Table update, Simple Query (2)	2446	279	278
Left Outer Join and Group By (2)	7045	276	264
Delete with Subquery (2)	4327	277	266
Correlated Subquery and Group By (4)	16217	535	508
Triggers (1)	3190	196	188
Stored Procedures and Functions (1)	6132	191	185

Table 2: Total number of submissions for each concept. This gives a general idea of how many submissions from each concept were in our data set, and sheds light on the fact that it often takes a student many attempts to complete a SQL homework problem.

Error (MySQL error code)	% of syntactic errors	% of times student encountered error and was successful on question
Syntax Error (1064)	48%	96%
Undefined Column (1054)	15%	96%
Summary Column not Included in Group By (1055)	6%	97%
Summary Column used without Group By (1140)	4%	97%
Column identifier is ambiguous (1052)	3%	98%
Undefined Table (1146)	3%	96%
Invalid use of aggregating function (1111)	2%	98%
Derived table must have own alias (1248)	2%	98%
Subquery returns more than 1 row (1242)	1%	99%
Duplicate Entry (1062)	1%	99%
Subquery returns more than 1 column (1241)	1%	97%
Expression in ORDER BY not in SELECT (3065)	1%	97%

Table 3: Breakdown of Syntactic errors.

column and errors related to GROUP BY, were also strong indicators of a student not being able to finish a problem successfully.

Table 4 shows the last errors that a student encountered before giving up on a problem, revealing that, contrary to prior results, students aren't always able to overcome syntax errors. Table 5 gives a breakdown of MySQL error codes based on the question the student was solving at the time, giving some insight into what was difficult about particular exercises. Another interesting observation from table 5 is that even though the solutions for triggers, functions, and stored procedures, are longer and arguably more complex than other queries, there isn't any particular type of error that students struggled with, they mostly ran into syntax errors, just like on any of the other exercises. Trends were very similar across the two semesters from which we pulled our data (Summer and Fall 2019).

6 Discussion

Table 1 shows that a substantial portion of these errors that were unrecoverable occur in questions that intend the student to use a

Error	Percent of failed final submissions (number)
Semantic Error	58% (75)
Syntax Error (1064)	27% (36)
Undefined Column (1054)	3% (4)
Undefined Table (1146)	3% (4)
Summary Column not Included in Group By (1055)	2% (3)
Invalid use of aggregating function (1111)	1% (2)

Table 4: Last Error before Student Gives up on Problem. More than half the time (58%) when student gave up on a problem, their final submission compiled, it just didn't give the correct result. Much of the time (27%), the student had a syntax error they were not able to resolve.

`GROUP BY` clause, or for problems requiring a correlated subquery. These findings for homework assessments align with observations from the related studies by Ahadi et al., Taipalus and Perälä, Reisner, and Welty and Stemple [1, 2, 19, 22, 23].

However, while Ahadi et al. and Taipalus and Perälä play down the impact of syntax errors, suggesting that they are not a major issue [1, 2, 21], we find that syntax errors are a non-trivial barrier stopping many students from completing their homework exercises. This might be due to the fact that a higher percentage of the homework problems we analyze cover more advanced topics than the problems analyzed by others. Students studied by Ahadi et al. and Taipalus and Perälä started out writing simple queries over a single table, and ended with correlated subqueries. On the other hand, the students we studied started with joins, and covered topics like left outer joins, triggers, and stored procedures which were not covered in any of the other studies. It seems when the queries get more complex, students are more likely to get stuck and never make it past their syntax errors (of the 36 final submissions which were syntax errors, 30 of them were from the last four concepts, which were not covered at all by prior studies).

6.1 Limitations

We consider only the outputted result, without taking the actual submitted query into consideration. While syntactic and semantic errors can be useful indicators of a lack of understanding, there is still valuable information left within submissions that are marked correct. Upon manual examination of some of these queries, we found many students were using more complex SQL features than what were needed to complete the problem they were working on, or take a different approach than the question intended. These end submissions might offer additional insights both to understanding student misconceptions Similarly, we consider each submission and its result as a single entity, without comparison to previous submissions. Considering this context could have also given further insight into students' thought processes.

Additionally, data was collected from only one school, potentially limiting the generalizability of findings. The SQL questions used for the submissions may not be reflective of other schools' curricula. Also, as mentioned earlier, the course covers several database topics in addition to SQL, therefore student behaviors observed may not be reflective of curriculum in which SQL is the only domain-specific language used.

The struggles of students working on small SQL examples for a university course also may not translate realistically to the troubles that people have in using SQL while working in industry. For example, we found students were essentially always able to solve problems containing natural joins, but this may not hold true if they are working with the larger data sets that are so common in industry.

Finally, students may have been less likely to finish some questions not because they were more difficult, but because they were the last SQL homework assignments for the class, and they might have decided that it was not worth their time to complete the final assignment based on the work they had completed so far, and the grade that they were content with.

6.2 Future Work

There are many available avenues for future work in database education. This and prior work analyzing student submission data gives good insight into the type of problems that are difficult for students when learning to query databases. One next step would be to conduct talk-aloud interviews with students to understand why they run into these problems, and the thought processes associated with them, so that instructors can improve the efficiency with which they address these issues.

Further analysis of the student submission data could also be enlightening. Jadud [11] introduced the idea of an "Error Quotient": a measurement of how difficult it is for students to overcome errors while programming, which has been used to give additional insight into the student problem solving process while programming in Java [11, 12]. Applying these methods could also give additional insight into how long students spend stuck on particular errors.

There is also a lack of tooling for database education in relation to other fields. Tools akin to Guo's Python tutor [9] could be extremely helpful in helping students to create a mental model of how a database engine executes their query, and a something like Glassmans' OverCode [8] could have an impact in helping instructors to diagnose student misconceptions in their classes early and often, especially when informed by the knowledge from this and other studies about which errors are most common and difficult to overcome.

Despite a growing body of literature on improving error messages for students learning to program [4, 5, 26] very little has been done to ensure that the error messages generated by database engines work well for helping students overcome the errors they face.

Finally, the recent rise in popularity of NoSQL and Graph databases like MongoDB [14] and Neo4j [15], respectively, present an entirely new challenge in education, with little to no extant research on how to best teach students how to use them.

7 Conclusion

Understanding student behavior in database education is still an under-explored area in Computer Science Education. We take a step in addressing this issue by analyzing student submissions

	Join and Group By	Join and Where	Join, Where, and Distinct	Table update, Simple Query	Left Outer Join and Group By	Delete with Subquery	Correlated Subquery and Group By	Triggers	Stored Procedures and Functions
Syntax Error (1064)	1313	914	301	403	1227	1346	2726	1842	2982
Undefined Column (1054)	537	293	156	224	603	453	1531	59	222
Summary Column not Included in Group By (1055)	372	78	29	80	412	16	672	2	58
Summary Column used without Group By (1140)	274	11	0	90	352	20	492	0	62
Column identifier is ambiguous (1052)	222	99	34	109	153	29	340	2	60
Undefined Table (1146)	128	80	40	68	89	97	324	73	58
Invalid use of aggregating function (1111)	279	12	0	1	51	275	110	3	4
Derived table must have own alias (1248)	136	67	23	18	75	39	185	0	16
Subquery returns more than 1 row (1242)	20	27	7	6	67	96	161	16	4
Duplicate Entry (1062)	50	36	23	34	27	31	94	6	67
Subquery returns more than 1 column (1241)	48	21	15	2	20	36	160	25	21
Expression in ORDER BY not in SELECT (3065)	8	6	5	0	1	0	257	0	0

Table 5: Number of submissions of most common syntactic errors per concept. Here we see again that the vast majority of syntactic errors are syntax errors, with the effect being even more pronounced for more compilicated topics. Other interesting patterns include the vast number of undefined column errors when students worked on left outer joins or correlated subqueries.

to 18 SQL homework problems for an upper-level database course. Our analysis shows that submissions resulting in syntactic errors were the majority. We echo prior work that errors relating to `GROUP BY` and correlated subqueries appear to be most difficult for students to overcome, and add the new finding that syntax errors can also be a non-trivial barrier for students to overcome, especially when writing more complex SQL queries. Using these categorizations on homework problems can be useful to instructors to gain a coarse understanding and address misconceptions before examination. Overall, this analysis serves as the next push towards new directions and better understanding in database education.

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] 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

[4] Brett A. Becker. 2016. An Effective Approach to Enhancing Compiler Error Messages. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16)*. ACM, New York, NY, USA, 126–131. https://doi.org/10.1145/2839509.2844584

[5] Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, Janice L. Pearce, and James Prather. 2019. Unexpected Tokens: A Review of Programming Error Messages and Design Guidelines for the Future. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '19)*. ACM, New York, NY, USA, 253–254. https://doi.org/10.1145/3304221.3325539

[6] 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

[7] Hector Garcia-Molina. 2008. *Database systems: the complete book*. Pearson Education India, Chennai, Tamil Nadu, India.

[8] Elena L. Glassman, Jeremy Scott, Rishabh Singh, Philip J. Guo, and Robert C. Miller. 2015. OverCode: Visualizing Variation in Student Solutions to Programming Problems at Scale. *ACM Trans. Comput.-Hum. Interact.* 22, 2, Article 7 (March 2015), 35 pages. https://doi.org/10.1145/2699751

[9] Philip J. Guo. 2013. Online Python Tutor: Embeddable Web-based Program Visualization for Cs Education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*. ACM, New York, NY, USA, 579–584. https://doi.org/10.1145/2445196.2445368

[10] Qiang Hao, David H. Smith IV, Naitra Iriumi, Michail Tsikerdekis, and Andrew J. Ko. 2019. A systematic investigation of replications in computing education research. *ACM Transactions on Computing Education* 19, 4, Article 42 (August 2019), 18 pages. https://doi.org/10.1145/3345328

[11] Matthew C. Jadud. 2006. Methods and Tools for Exploring Novice Compilation Behaviour. In *Proceedings of the Second International Workshop on Computing Education Research (ICER '06)*. ACM, New York, NY, USA, 73–84. https://doi.org/10.1145/1151588.1151600

[12] Matthew C. Jadud and Brian Dorn. 2015. Aggregate Compilation Behavior: Findings and Implications from 27,698 Users. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research (ICER '15)*. ACM, New York, NY, USA, 131–139. https://doi.org/10.1145/2787622.2787718

[13] A. Mitrovic. 2003. An intelligent SQL tutor on the web. *International Journal of Artificial Intelligence in Education* 13, 2-4 (2003), 173–197. cited By 182.

[14] MongoDB, Inc. 2019. MongoDB. https://www.mongodb.com/

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

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

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

[18] Jay Patel. 2017. The 9 Most In-Demand Programming Languages of 2017. https://www.codingdojo.com/blog/9-most-in-demand-programming-languages-of-2017 [Online; accessed 10-January-2020].

[19] Phyllis Reisner. 1977. Use of Psychological Experimentation as an Aid to Development of a Query Language. *IEEE Transactions on Software Engineering* SE-3, 3 (May 1977), 218–229. https://doi.org/10.1109/TSE.1977.231131

[20] 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

[21] 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 (SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 198–203. https://doi.org/10.1145/3287324.3287359

[22] 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

[23] Charles Welty and David W. Stemple. 1981. Human Factors Comparison of a Procedural and a Nonprocedural Query Language. *ACM Trans. Database Syst.* 6, 4 (Dec. 1981), 626–649. https://doi.org/10.1145/319628.319656

[24] 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.

[25] Craig Wills. 2020. Analysis of Current and Future Computer Science Needs via Advertised Faculty Searches for 2020. https://cra.org/analysis-of-current-and-future-computer-science-needs-via-advertised-faculty-searches-for-2020/ [Online; accessed 13-January-2020].

[26] John Wrenn and Shriram Krishnamurthi. 2017. Error Messages Are Classifiers: A Process to Design and Evaluate Error Messages. In *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2017)*. ACM, New York, NY, USA, 134–147. https://doi.org/10.1145/3133850.3133862