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

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 particularly hard time constructing queries requiring correlated subqueries or GROUP BY clauses. Ahadi et al. [1] and Welty and Stemple [23] both also point out the 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 correct answer (i.e., the set of results the query should return) and 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)
GROUP BY A.BrandName
```

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
### 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.

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

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.

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

### Table 3: Breakdown of Syntactic errors.

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

### 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 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).
We consider only the outputted result, without taking the actual what were needed to complete the problem they were working on, which were not covered at all by prior studies).

We found many students were using more complex SQL features than the submissions may not be reflective of other schools’ curricula. 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, still valuable information left within submissions that are marked correct. Much of the time (27%), the student had a syntax error they were not able to resolve.

6.1 Limitations

We consider only the outputted result, without taking the actual submitted query into consideration. 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 Neo[4] [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...
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


