Explaining Causes Behind SQL Query Formulation Errors

Toni Taipalus
Faculty of Information Technology
University of Jyvaskyla
Jyvaskyla, Finland
https://orcid.org/0000-0003-4060-3431

***Abstract*—This Full Research Paper presents the most prominent query formulation errors in Structured Query Language (SQL), and maps these errors to their cognitive explanations. Understanding query formulation errors is a key to teaching SQL. more effectively. However, studies on what kind of errors novices struggle with are relatively scarce when compared to, for example, programming languages. Although committing errors is a crucial part in learning, some errors are relatively easy to fix, and their commonness is not necessarily an indication of their difficulty. Other errors, however, halt the learning process, and are never fixed by the query writer. Using a previously established error taxonomy and queries from four cohorts with a total of 987 students, we set out to identify common errors which students are unable to correct, i.e., errors that are likely to cause query formulation failures. Our results indicate that on a general level, logical errors are the most common cause for query formulation failures, while syntax and semantic errors are usually fixed by query writers. Although query concepts, for example, expressions, joins and grouping, have a strong influence on what types of errors are committed, some errors are common regardless of query concepts. Specifically, our results indicate that missing expressions, extraneous or omitted grouping columns, incorrect comparison operators, missing joins, and missing ordering columns are the most common errors that novices are unable to fix. Based on the results, we speculate on the reasons behind the most common persistent errors using previously identified cognitive explanations. Finally, we present that solutions for mitigating the causes behind query formulation errors are already available. In order to more effectively teach query formulation, educators should emphasize natural language patterns, query planning, and increasingly ambiguous exercises.**

***Index Terms*—Structured Query Language (SQL); database; error; education; novice**

I. Introduction

Structured Query Language (SQL) is widely taught in higher education, and recommended in computer science [1], software engineering [2] and information systems [3] curricula guidelines. In industry, SQL remains the *de facto* language to query data from databases. Even though programming languages have received ample attention in educational research [4], SQL, in comparison, has largely remained in the sidelines [5]. Although it can be argued that studies concerning programming languages can be generalized to cover SQL as well, SQL is not a programming language, and the paradigm behind SQL is declarative, as opposed to the imperative nature of many programming languages studied [6].

Novices to any computer language commit various different errors when learning a language, and understanding the nature of these errors is crucial to understanding the learning process as a whole. In this regard, educational SQL research has only in recent years started to systematically study the errors that novices commit [7] [6] [8] [9] [10]. Even so, many of such studies did not study errors per se, but rather used *a posteriori* error classification as a vehicle for answering their research questions. Furthermore, as we are beginning to understand *what* errors novices commit, we are able to move towards *why* these errors occur, and start to mitigate them. However, research explaining causes behind errors is both scarce, and in some cases, possibly obsolete [5] due to additional features introduced in the SQL language. To that end, we conducted a quantitative study to find out which *persistent* (i.e., never corrected) errors are most common in certain types of queries, and mapped these errors to cognitive explanations introduced previously [11].

The rest of this study is structured as follows. Next, in Section II we describe prior work related to SQL education, key terms, and query formulation errors and their explanations. In Section III we describe how our data was collected, our study participants, and show our contribution *vis-a-vis* prior work. In Section IV we present both the frequencies of the most common errors, and map these errors to their respective causes. In Section V to discuss the practical implications of our study, and threats to validity. Finally, in Section VI we present conclusions.

II. Background

A. SQL in Educational Research

In terms of educational research, prevalent SQL research topics are different learning environments [12] [13] [14], various teaching approach propositions [15] [16] [17], and understanding novice errors [8] [9]. Although SQL is a versatile language, both SQL education and research usually focus on data retrieval, i.e., SELECT statements [5]. Much of the research seem to agree that students should learn SQL in practice in addition to learning theoretical foundations [18] [19].

The learning environments that enable and facilitate SQL learning are aplenty, and differ in terms of maturity and amount of features. What is mutual in these environments

is that they usually present the user (i.e., a student) with the current *data demand* in natural language (e.g., "List all customers who have ordered an item in 2019"), and the database structure. The student is then expected to write an SQL equivalent for the given data demand, submit the query to the database management system, and receive the correct result table. If the database management system outputs an error message instead of a result table, or the result table is incorrect for the given data demand, the student has committed a query formulation error.

B. Query Formulation Errors

Commonly, query formulation errors have been divided into two classes: *syntax* and *semantic* errors [11] [20], arguably because database management systems can detect the former. More recently, it has been suggested that there is a subset of semantic errors which can, in theory, be detected by the database management systems even if the data demand is unknown, because queries that contain such errors are "always wrong" [7]. The remaining subset of semantic errors which can only be detected if the current data demand is known is called *logical* errors [10]. Additionally, *complications*, i.e., unnecessary elements which do not affect the correctness of the result table, should be considered as well [7]. An error categorization framework was constructed around these four classes of errors [10]. The framework is effectively a three level taxonomy of 105 different errors (Fig. 1).

A key thing to remember regarding SQL errors is that there is no single stand to *what SQL really is*. The language is defined by the SQL standard [21] [22], but different database management systems implement SQL differently [23], and what is a syntax error to, e.g., Oracle Database may not be a syntax error to SQL Server. This problem concretizes as students sometimes use a single database management system when learning SQL. Educational research has only very recently attempted to address the problem of database management system dependency in error categorization [10].

Successfully formulating an SQL query has been shown to be related to the query concepts (e.g., joins, ordering, expressions, and grouping) [6], and certain query concepts have been shown to invite certain errors [24]. Furthermore, both syntax and logical errors have been argued to be the main cause of query formulation failures, and current evidence on the topic remain under debate. In the center of the discussion appears to be the equivalence of errors committed during query formulation versus errors that are never corrected, i.e., persistent errors. While some studies have examined error frequencies during the whole query formulation process [8] [10], some studies argue that not all errors are equal. While writing erroneous queries is a natural and beneficial part of the learning process, an error which is never fixed is an indication of a problem. For example, syntax errors are frequent in query formulation, but usually fixed, while logical errors are similarly frequent, but students are less likely successful in fixing them [10] [24].

C. Explanations Behind Errors

Explanations behind errors can be divided into causes stemming from the learner, and causes stemming from the environment. Regarding environmental causes, it has been shown that, e.g., data demand ambiguity [25] [26] [27], database schema representation [28] [29], query complexity [26], and how well real-world constructs match their equivalents in the database [26] all affect the number of errors committed. While these aspects are important, in this study we focus on causes stemming from the learner.

Attempts have been made to explain causes behind query formulation errors [30] [31] [20] [32]. As the first version of the SQL standard was released in 1986, all these categorizations were constructed when the language was in a nascent stage. More recently, and in the spirit of Reisner's [33] [30] seminal studies, Smelcer [11] discussed causes behind different errors with cognitive explanations. Even more recently, Ahadi et al. [9] mapped 551 erroneous SQL queries regarding selected query concepts to the four aforementioned categorizations from the 1970s and 1980s. To our knowledge, these studies are the sole attempts which have tried to uncover the root causes behind query formulation errors according to a previously established framework. In contrast, several studies on SQL errors have speculated causes in their respective discussion sections.

According to Smelcer [11], there are four root causes for query formulation errors. First, (human) *working memory overload* causes errors of omission in particular. As the number of joins and expressions in a data demand increase, some items are displaced from working memory, and never written in the corresponding query. Second, *absence of retrieval cue* is closely related to the data demand. Some data demands contain more cues or hints about how the corresponding query should be formulated. However, some data demands lack such cues. For example, a data demand "Find all US customers who have ordered an item in 2020" contains cues for expressions regarding nationality and order year, but lacks cues regarding the fact that the query possibly requires joins, as customer and order details are likely stored in different tables. Third, *procedural fixedness* refers to query writers becoming accustomed to writing certain kinds of queries. This may become a problem when a subsequent query requires different kind of formulation, but the writer is fixed on solving the problem with previously used techniques. Finally, *absence of procedural knowledge* (i.e., simply *ignorance*) is effectively insufficient or incorrect knowledge about how SQL or the relational model works.

III. Research Setting

A. Data Collection

We collected the data from four student cohorts over a timeframe of four years. The course was an introductory database course targeted for second year university students majoring in computer science or information systems, with no prior knowledge on SQL. The course consisted of lectures,

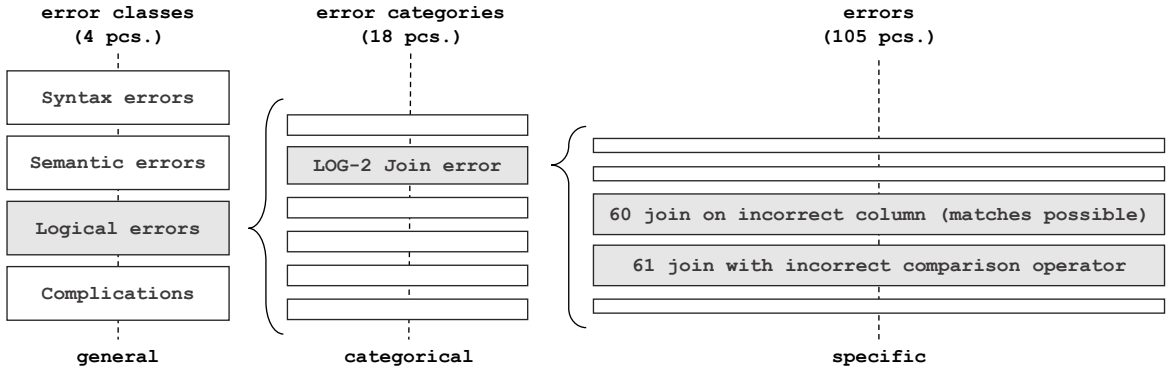

Fig. 1. Error taxonomy [10]: from general to specific level, 4 classes of errors are divided into 18 categories, which are further divided into 105 different errors - only a small subset of errors and error categories is illustrated here

voluntary exercises, discussion of said exercises, and an exam. Concerning data retrieval, exercises were designed using a previously established query concept framework (Table I). The 15 exercises were divided into three sets A, B, and C, and the students had approximately one week to complete exercises in a set. The students were given unlimited number of attempts within the timeframe, and they could use whatever materials or ways of communication at their disposal. The exercises within a set could be attempted in any order, and the the correct result table and the database schema diagram were visible during the whole query writing process. The students were able inspect the table creation statements with built-in commands, and the learning environment was effectively an SQLite command prompt embedded into a web page. Once a weekly deadline had passed, students were presented with example answers, and an opportunity to discuss alternative methods of formulating the correct queries. Each cohort had a similar set of exercises, but with a database using a different business domain. Relevant query concepts are discussed and demonstrated in detail in Section IV.

TABLE I
QUERY CONCEPTS FOR EACH EXERCISE, BASED ON TAIPALUS ET AL. [10]

Exercise	Concepts
A1	single-table; expressions
A2	single-table; expressions; ordering
A3	single-table; wildcard; expressions with nesting
B4	multi-table; expressions; facing foreign keys
B5	multi-table; expressions; ordering
B6	multi-table; expressions with nesting; ordering
B7	multi-table; expressions; does not exist
B8	multi-table; does not exist; equal subqueries
B9	single-table; expressions; aggregate functions
B10	multi-table; expressions; multiple source tables
B11	multi-table; expressions; self-join; aggregate function evaluated against a column value; uncorrelated subquery
B12	multi-table; expressions; aggregate function evaluated against a constant; correlated subquery; parameter distinct
B13	multi-table; expressions; self-join
C14	multi-table; multiple source tables; aggregate functions; grouping
C15	multi-table; multiple source tables; aggregate functions; grouping; grouping restrictions; ordering

B. Method

Four student cohorts with a total of 987 students yielded over 176,000 SQL queries, out of which we selected only the chronologically last query for each student for each exercise (i.e., final queries) for analysis. We coded these 12,180 final queries according to the error categorization framework [10], and, if a query contained at least one syntax, semantic, or logical error, the query was considered incorrect. We also coded complications, but did not regard them in this study. 73 students committed no persistent errors, so finally, we were left with 3,739 incorrect final queries from 914 students which we selected for further analysis.

Out of the numerous concepts of the query concept framework (cf. Table I), we selected only the ones highlighted in a previous study [24]: multi-table (the query must be formulated using multiple tables rather than only one, exercises B4..B8 and B10..C15), equal subqueries (subqueries are on the same level with each other rather than nested, exercise B8), self-join (a table is joined with itself, exercises B11 and B13), multiple source tables (the result table contains columns from multiple tables rather than only one, exercises B10, C14 and C15), and aggregate functions (the query must be formulated using an aggregate function, exercises B9, B11, B12, C14 and C15). For each of these concepts, we selected final queries for corresponding exercises, and counted the frequencies of errors.

C. Relationship with Previous Work

We utilized previously reported query concepts [10] in our exercise design, and categorized student errors according to an error categorization framework [10]. The data collected yielded SQL queries regarding 19 query concepts, out of which five were selected for further analysis. Specifically, these five concepts were selected because they have been previously shown to invite more clearly distinguishable errors than other concepts [24]. Next, the discovered most common persistent errors for each of these five query concepts were mapped to Smelcer's [11] four cognitive explanations. In the mapping process, we considered the order in which the queries were

presented to students, query complexity, and query concepts. Similarly to Smelcer [11], the mapping was speculative.

Now that we have elaborated our research setting and studies influencing it, it is natural to discuss our main contributions in relation to prior studies. Regarding the discovery of the most common persistent errors, Taipalus and Perälä [24] had a similar research question to us, and they used the same error categorization framework. However, their unit of observation was *error categories*, which all contain more than one error. We, however, use *errors* as the unit of observation, which allow more fine-grained insights. Regarding causes of errors, both Smelcer [11] and Ahadi et al. [9] had a similar research question.

IV. The Most Common Errors and Their Causes

In this section, we list the most common persistent query formulation errors, and map the most common errors within certain query concepts to Smelcer's [11] cognitive explanations: working memory overload, absence of retrieval cue, procedural fixedness, and ignorance. Furthermore, we speculate *why* certain explanations fit certain errors. We also provide example queries which demonstrate the query concepts discussed in the following subsections. The example queries correspond to the following database schema. The database schema is not the same as used in the research setting, but a minimal example for clarity and brevity:

- CUSTOMER(customerid, name, address)
- ORDERED (productid, customerid, quantity)
- PRODUCT (productid, description, onhand)
- SUPPLIES (supplierid, productid)
- SUPPLIER (supplierid, name)

A. All Concepts

Out of the total 12,180 final queries, 3,739 were incorrect (i.e., contained at least one syntax, semantic or logical error). The most common errors in all queries, regardless of the query concept, were missing expressions (a logical error) with the frequency of 0.29, extraneous or omitted grouping column (a syntax error) with the frequency of 0.18, as well as missing join, and missing column from the ORDER BY clause (both logical errors), both with the frequency of 0.10. The most common persistent errors across all queries are listed in Table II. The most common persistent errors across different query concepts (multi-table, equal subqueries, self-join, multiple source tables, aggregate functions) are illustrated in Fig. 2. In the spirit of Smelcer [11], we consider errors caused by ignorance theoretically uninteresting, and do not discuss them in as much detail as the other three causes.

Concerning Fig. 2, it is worth noting that out of the 3,739 incorrect final queries, many were analyzed multiple times, as one query can test student skill concerning, e.g., multi-table queries and aggregate functions. In fact, it is impossible for a query to test student skill with multiple source tables without testing skill with multi-table queries. Additionally, an incorrect query can exhibit more than one error, and only the six most frequent errors for each query concept are reported.

TABLE II
Error frequencies of the six most common persistent errors across all queries

Error ID and description	Freq.
missing expression	0.29
extraneous or omitted grouping column	0.18
missing join	0.10
missing column from ORDER BY clause	0.10
incorrect comparison. op. or incorrect value compared	0.08
omitting a join	0.06
number of queries analyzed	*3,739*

TABLE III
Error frequencies of the six most common persistent across multi-table queries

Error ID and description	Freq.	Cause
missing expression	0.31	memory overload
extraneous or omitted grouping column	0.22	ignorance
missing join	0.12	absence of cue
incorrect comparison op. or value compared	0.08	ignorance
omitting a join	0.07	ignorance
missing column from SELECT	0.06	ignorance
number of queries analyzed	*2,869*	

Effectively, this means that the x-axis in Fig. 2 adds up to more than 1, and that the length of a bar merely represents the sum of the top six error frequencies within a concept.

B. Multi-table

As can be observed in Table III, our results indicate that across multi-table queries, missing expressions and grouping related errors are more common than join errors. The used error categorization framework [10] divides absent joins into *omitting a join* and *missing join*. The former is a semantic and the latter a logical error. A prime example of the former would be the query in (1) without the last line (i.e., the query outputs a Cartesian product), and a prime example of the latter in (2) without the last two lines.

```
SELECT c.name
FROM   customer c, ordered o
WHERE  c.customerid = o.customerid;
```

Listing 1. A multi-table query with an explicit join condition

```
SELECT c.name
FROM   customer c
JOIN   ordered o
  ON   (c.customerid = o.customerid);
```

Listing 2. ANSI SQL-92 join in a multi-table query

Although Smelcer [11] indicates that "omitting the join clause" is the most common user error, there are two things worth noting. First, Smelcer [11] demonstrates joins using an explicit join condition in the WHERE clause, similar to (1). This alone supports the assumption that Smelcer [11] uses the term *join clause* for the explicit join, rather than the keyword JOIN. Second, if joins are formulated as in (1), omitting a join (i.e., the WHERE clause) is arguably more understandable than omitting the last two lines of (2). This

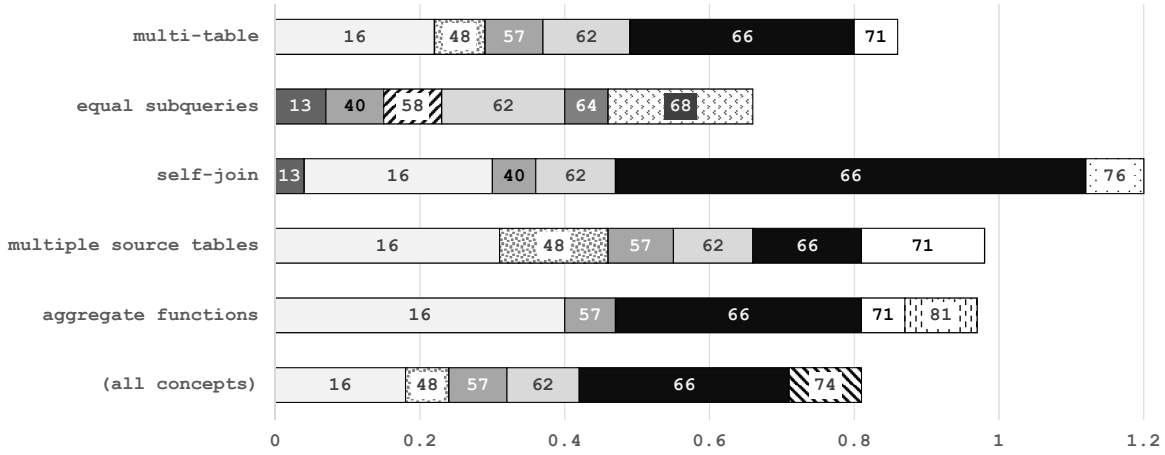

Fig. 2. The six most frequent persistent errors by query concept. The bar labels refer to the error IDs [10]: 13 = data type mismatch, 16 = grouping error: extraneous or omitted grouping column, 40 = implied, tautological or inconsistent expression, 48 = omitting a join, 57 = incorrect comparison operator or incorrect value compared, 58 = join on incorrect table, 62 = missing join, 64 = improper nesting of subqueries, 66 = missing expression, 68 = extraneous expression, 71 = missing column from SELECT, 74 = missing column from ORDER BY clause, 76 = extraneous ORDER BY clause, 81 = incorrect column as function parameter

TABLE IV
ERROR FREQUENCIES OF THE SIX MOST COMMON PERSISTENT ERRORS ACROSS QUERIES WITH EQUAL SUBQUERIES

Error ID and description	Freq.	Cause
extraneous expression	0.21	ignorance
missing join	0.17	absence of cue
implied, tautological or inconsistent expr.	0.08	ignorance
join on incorrect table	0.08	ignorance
data type mismatch	0.07	ignorance
improper nesting of subqueries	0.07	procedural fixedness
number of queries analyzed	*156*	

observation propounds the view that the method used for formulating joins in Smelcer's [11] research may have been a factor that increased join omissions. Missing expressions were common in multi-table queries, which supports Smelcer's [11] observations.

C. Equal Subqueries

Writing equal subqueries (3) after its counterpart, nested subqueries, is a prime example of Smelcer's [11] concept of *procedural fixedness*. As the participants were accustomed to writing nested subqueries in four preceeding exercises, using nested subqueries was arguably an intuitive strategy for this exercise as well. However, nesting the subqueries, illustrated in (4) is semantically correct, but answers to a different data demand than (3).

```
SELECT p.productid, p.description
FROM   product p
WHERE  NOT EXISTS
  (SELECT *
   FROM   ordered o
   WHERE  p.productid = o.productid)
AND    EXISTS
  (SELECT *
   FROM   supplies s
   WHERE  p.productid = s.productid);
```

Listing 3. A query with equal subqueries

```
SELECT p.productid, p.description
FROM   product p
WHERE  NOT EXISTS
  (SELECT *
   FROM   ordered o
   WHERE  p.productid = o.productid
   AND    EXISTS
      (SELECT *
       FROM   supplies s
       WHERE  p.productid = s.productid)
  );
```

Listing 4. A query with nested subqueries

Rather surprisingly, the most common error in the equal subquery exercise was extraneous expression (Table IV). A possible explanation is that students first formulated nested subqueries, and, as the nested subqueries example in (3) always returns more data, students may have tried to eliminate result table rows by adding expressions. In this case, however, such expressions are extraneous. While data demands usually contain cues for expressions, cues for joins are seldom present. Intuitively, missing joins can be associated with the absence of cues.

D. Self-join

For novices, self-join is one of the most difficult query concepts [6] [8] [9] [24]. Self-join queries are also prime examples of Smelcer's [11] *absence of retrieval cue*, as the data demands of such queries do not contain a cue for a table

TABLE V
ERROR FREQUENCIES OF THE SIX MOST COMMON PERSISTENT ACROSS SELF-JOIN QUERIES

Error ID and description	Freq.	Cause
missing expression	0.65	absence of cue
extraneous or omitted grouping column	0.26	ignorance
missing join	0.11	absence of cue
extraneous ORDER BY clause	0.08	ignorance
data type mismatch	0.04	ignorance
implied, tautological or inconsistent expr.	0.04	ignorance
number of queries analyzed	*750*	

TABLE VI
ERROR FREQUENCIES OF THE SIX MOST COMMON PERSISTENT ERRORS ACROSS QUERIES WITH MULTIPLE SOURCE TABLES

Error ID and description	Freq.	Cause
extraneous or omitted grouping column	0.31	ignorance
missing column from SELECT	0.17	ignorance
omitting a join	0.15	ignorance
missing expression	0.15	memory overload
missing join	0.11	ignorance
incorrect comparison op. or value compared	0.08	ignorance
number of queries analyzed	*807*	

TABLE VII
ERROR FREQUENCIES OF THE SIX MOST COMMON PERSISTENT ERRORS ACROSS QUERIES WITH AGGREGATE FUNCTIONS

Error ID and description	Freq.	Cause
extraneous or omitted grouping column	0.40	ignorance
missing expression	0.34	absence of cue
incorrect column as function parameter	0.10	ignorance
incorrect comparison op. or value compared	0.07	ignorance
extraneous ORDER BY clause	0.06	ignorance
missing column from SELECT	0.06	ignorance
number of queries analyzed	*1,486*	

join. In fact, a data demand "Find the names of customers who live in the same address as customer #47" (5) might, to a novice, even seem to discourage joining tables. Furthermore, the data demand for the query in (5) typically omits the required expression on the fifth line. Finally, the data demand for the query in (6) omits the requirement that product description must be checked twice. For these reasons, we speculate that in the case of self-join, missing expressions are not caused by working memory overload, but absence of retrieval cues (Table V). However, it can be argued that a missing expression in the query in (6) is caused by inaccurate procedural knowledge, as writing both expressions presupposes knowledge about what the uncorrelated subquery will return during execution.

```
SELECT c1.name
FROM   customer c1
JOIN   customer c2
  ON   (c1.address = c2.address)
WHERE  c1.customerid <> 47
  AND  c2.customerid = 47;
```

Listing 5. A typical, simple self-join

```
SELECT productid, onhand
FROM   product
WHERE  description = 'used'
  AND  onhand =
     (SELECT MAX(onhand)
      FROM   product
      WHERE  description = 'used');
```

Listing 6. A self-join with an uncorrelated subquery formed by evaluating an aggregate function against a column value

E. Multiple Source Tables

A query with multiple source tables projects or calculates column values into the result table from more than one table (7). In our exercises, this query concept was tested first in exercise B10, meaning that students formulated queries with a single source table in the preceeding nine exercises. The most common errors for this type of queries, however, do not list errors we could relate to procedural fixedness (Table VI).

```
SELECT o.productid, c.name, c.address
FROM   ordered o
JOIN   customer c
  ON   (o.customerid = c.customerid);
```

Listing 7. An example of a query with multiple source tables

We believe that the key to successfully writing a basic query with multiple source tables rests within the method of writing joins. If joins are written like in (7) or in (1), the query formulation process is relatively easy. If, however, a student formulates joins with EXISTS (3), scope related problems arise. Consequently, attempts to fix these problems while insisting on the use of EXISTS (or alternatively, IN) leads to a number of different and diverse errors, as can be observed in Table VI.

F. Aggregate Functions

Aggregate functions often imply grouping (8),(9), and grouping is a difficult concept [34]. As we have listed in Table VII, most aggregate function related errors are caused by ignorance, i.e., inaccurate procedural knowledge about how the language operates. Again, we believe that missing expressions are caused by absence of a cue rather than working memory overload (6).

```
SELECT   productid, SUM(quantity)
FROM     ordered
GROUP BY productid;
```

Listing 8. A query with an aggregate function and grouping

```
SELECT   p.productid
       , SUM(o.quantity)
       , COUNT(DISTINCT o.customerid)
FROM     product p, ordered o
WHERE    p.productid = o.productid
GROUP BY p.productid;
```

Listing 9. A query with aggregate functions, parameter distinct, multiple source tables and grouping

V. DISCUSSION

A. A Comparison of Results

As we reported previously, the studies of Smelcer [11] and Ahadi et al. [9] are closely related to ours. Overall, a one–to–one comparison of results is impossible due to the different

error categorizations, and in the case of Ahadi et al. [9], errors were also mapped to different causes. Furthermore, in the case of Smelcer [11], different types of SQL joins pose a threat to reliable comparison. Therefore, all comparison is subject to speculation.

The reported error frequencies are partly in line with Taipalus and Perälä [24], but provide much needed low level insight on the frequencies of *errors*, as opposed to *error categories* (cf. Fig. 1). As error categories contain several errors, some of these errors might be persistent in one category while others might not. Therefore, simply reporting the most frequent error categories might not convey which errors are related to particular query concepts. Two patterns regarding the most frequent errors can be observed in Fig. 2. Missing expression seems to be the most frequent persistent error throughout all exercises, as well as in many of the concepts. This result is in line with previously reported findings [9]. The next most frequent error appears to be extraneous or omitted grouping column, and the difficulties around the concept of grouping in SQL have been speculated even before the release of the SQL-86 standard [34], and evidence supporting this view presented in Ahadi et al. [9]. Our most notable extension to Smelcer's [11] work is that we present that cognitive explanations are not merely related to errors, but also to query concepts. For example, we presented that, depending on the context, missing expressions may be cause by either memory overload or the absence of retrieval cues.

B. How to Mitigate Persistent Errors

We presented the most common persistent query formulation errors and speculated on their cognitive explanations. Here we present how knowledge on the most common errors can be utilized in mitigating them. We also present that there already exist solution proposals in scientific literature which can be used to mitigate the root causes behind query formulation errors.

The error frequencies provide needed fine-grained information on which errors are likely never corrected by SQL novices. These findings can be used in classroom to emphasize difficult concepts such as comparison operators, projection, joins, and grouping. Furthermore, as discussed in Taipalus et al. [10], the most frequent errors may be taken into account when designing exercise databases and exercise data. If the correct result table is presented during query writing, and the exercise data is designed so that student queries with the most frequent errors output a different result table, the students are made aware that their query is incorrect. Several studies have proposed teaching SQL by showing students incorrect SQL queries [35] [15] [36]. Rather than formulating all possible or an arbitrary set of erroneous queries, we suggest that educators utilize the most common persistent errors identified in this study. Because this study only considered persistent errors, the most frequent errors reported may be interpreted as the most common causes of failure in query formulation.

In addition to SQL in particular, working memory capacity has received increasing attention in database education in general [37]. If educators were to strive to increase query formulation success rates, formulating simple exercises to mitigate working memory load would be an intuitive solution. However, as education aims to train future professionals, formulating only simple exercises is not a feasible solution. Given the centrality of the issue of cognitive load, some solutions have already been proposed in the form of simpler database structures [38], query templates [39] [40] and a planning notation for complex SQL queries [41]. Query planning, in particular, arguably mitigates working memory constraints as relevant parts of the data demand are separated from irrelevant, and explicitly marked as a query plan which is then written in SQL.

On the subject of retrieval cues, the same argument for training professionals applies. In industry, data demands are often ambiguous [42], and students should be prepared to work with ambiguous data demands. An ambiguous data demand intuitively lacks retrieval cues, thus possibly increasing related errors. However, it has been shown that as data demand complexity increases, the numbers of query formulation errors in ambiguous and unambiguous data demands converge [27], i.e., ambiguity has less and less an effect on the number of errors with more and more complex data demands. Therefore, it seems justified to suggest that in education, simple data demands should be presented unambiguously, and as data demand complexity in consecutive exercises increases, so should ambiguity.

Finally, procedural fixedness is related to habits or routines in query formulation. Again, a rather intuitive but counterproductive approach would be to reveal the query concepts related to an exercise to a student prior to query writing, thus possibly mitigating errors caused by procedural fixedness. Rather, educators should teach students to recognize patterns in the natural language data demand, and map these patterns to corresponding SQL constructs. Such approaches have been presented [43] [44], and we propose that they are a natural fit to counter procedural fixedness. In summary, it seems justified to argue that most of the causes of errors could be mitigated by already presented teaching approaches. However, rather than in isolation, these approaches should be utilized in tandem, as they are not mutually exclusive.

C. Threats to Validity

There are several threats to validity possibly affecting the results. First, it is worth noting that the amount of query concepts potentially affect the error frequencies, e.g., if more of the 15 exercises tested student skill with single-table rather than multi-table queries, it is safe to state that the frequencies of join related errors would decrease. Second, the order in which the exercises are presented to students possibly affect error frequencies, e.g., if a self-join was tested later rather than earlier, the frequencies of errors unrelated to self-join would likely decrease, as students would have grown more accustomed to other query concepts in exercises they have completed earlier. Third, the used query concept framework is presented in a high level of detail. Effectively, this means

that a query that is complex enough to be pedagogically interesting, represents numerous query concepts. Consequently, the amount and nature of auxiliary query concepts affect the error frequencies for each query concept. Rather than trying to control these variables, we chose to study student learning in a more natural setting. Although this approach introduced several threats to validity, we believe they can be partly justified by the upsides, which are twofold. First, we believe a less controlled setting to be a major cause regarding the number of participants, which, in turn, mitigates risks associated with small sample sizes. Second, we have studied student learning in its natural environment, rather than in a more controlled lab setting, which naturally would have introduced different threats.

VI. Conclusion

This study was an attempt to provide a fine-grained analysis on which SQL errors are likely to cause novices to fail in SQL query formulation, and to explain causes behind most common errors. In all tested exercises, the three most frequent persistent errors were missing expressions, extraneous or omitted grouping columns, and missing joins. Furthermore, it was observed that the cause of error does not rest solely on the type of error committed, but also on the query concept. Finally, we proposed that three of the four most common causes for query formulation errors may be mitigated by teaching students to recognize natural language patterns and their SQL equivalents, *a priori* query planning, and gradually moving towards more and more ambiguous data demands.

References

[1] Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society, "Computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science," New York, NY, USA, Tech. Rep., 2013, 999133. [Online]. Available: doi.org/10.1145/2534860

[2] The Joint Task Force on Computing Curricula, "Curriculum guidelines for undergraduate degree programs in software engineering," New York, NY, USA, Tech. Rep., 2015. [Online]. Available: https://dl.acm.org/citation.cfm?id=2965631

[3] H. Topi, K. M. Kaiser, J. C. Sipior, J. S. Valacich, J. F. Nunamaker, Jr., G. J. de Vreede, and R. Wright, "Curriculum guidelines for undergraduate degree programs in information systems," New York, NY, USA, Tech. Rep., 2010. [Online]. Available: https://dl.acm.org/citation.cfm?id=2593310

[4] A. Ko and B. Myers, "Development and evaluation of a model of programming errors," in *IEEE Symposium on Human Centric Computing Languages and Environments*. IEEE, 2003. [Online]. Available: https://doi.org/10.1109%2Fhcc.2003.1260196

[5] T. Taipalus and V. Seppänen, "SQL education: A systematic mapping study and future research agenda," *ACM Transactions on Computing Education*, (in press). [Online]. Available: https://doi.org/10.1145/3398377

[6] A. Ahadi, J. Prior, V. Behbood, and R. Lister, "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)*. New York, New York, USA: ACM Press, 2015, pp. 201–206. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2729094.2742620

[7] S. Brass and C. Goldberg, "Semantic errors in SQL queries: A quite complete list," *Journal of Systems and Software*, vol. 79, no. 5, pp. 630–644, May 2006. [Online]. Available: https://doi.org/10.1016%2Fj.jss.2005.06.028

[8] A. Ahadi, V. Behbood, A. Vihavainen, J. Prior, and R. Lister, "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)*. New York, New York, USA: ACM Press, 2016, pp. 401–406. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2839509.2844640

[9] A. Ahadi, J. Prior, V. Behbood, and R. Lister, "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)*. New York, New York, USA: ACM Press, 2016, pp. 272–277.

[10] T. Taipalus, M. Siponen, and T. Vartiainen, "Errors and complications in SQL query formulation," *ACM Transactions on Computing Education*, vol. 18, no. 3, pp. 15:1–15:29, August 2018. [Online]. Available: http://doi.acm.org/10.1145/3231712

[11] J. B. Smelcer, "User errors in database query composition," *International Journal of Human-Computer Studies*, vol. 42, no. 4, pp. 353–381, April 1995. [Online]. Available: https://doi.org/10.1006%2Fijhc.1995.1017

[12] K. Atchariyachanvanich, S. Nalintippayawong, and T. Julavanich, "Reverse SQL question generation algorithm in the DBLearn adaptive e-learning system," *IEEE Access*, vol. 7, pp. 54 993–55 004, 2019.

[13] A. Mitrovic and S. Ohlsson, "Implementing CBM: SQL-Tutor after fifteen years," *International Journal of Artificial Intelligence Education*, vol. 26, no. 1, pp. 150–159, March 2016.

[14] A. Hull and B. du Boulay, "Motivational and metacognitive feedback in SQL-tutor*," *Computer Science Education*, vol. 25, no. 2, pp. 238–256, April 2015. [Online]. Available: https://doi.org/10.1080%2F08993408.2015.1033143

[15] C. Myers and P. Douglas, "The un-structured student," in *24th British National Conference on Databases (BNCOD)*, July 2007, pp. 3–9.

[16] L. I. McCann, "On making relational division comprehensible," in *Proceedings of the 2003 33rd Annual Frontiers in Education Conference (FIE)*, vol. 2, November 2003, pp. F2C–6.

[17] A. Fekete, "Teaching transaction management with SQL examples," in *Proceedings of the 2005 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, ser. ITiCSE '05. New York, NY, USA: ACM, 2005, pp. 163–167. [Online]. Available: http://doi.acm.org/10.1145/1067445.1067492

[18] C. Pahl, R. Barrett, and C. Kenny, "Supporting active database learning and training through interactive multimedia," in *Proceedings of the 2004 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, ser. ITiCSE '04. New York, NY, USA: ACM, 2004, pp. 27–31. [Online]. Available: http://doi.acm.org/10.1145/1007996.1008007

[19] M. K. S. Sastry, "An effective approach for teaching database course," *International Journal of Learning, Teaching and Educational Research*, vol. 12, no. 1, 2015.

[20] C. Welty, "Correcting user errors in SQL," *International Journal of Man-Machine Studies*, vol. 22, no. 4, pp. 463–477, April 1985. [Online]. Available: https://doi.org/10.1016%2Fs0020-7373%2885%2980051-1

[21] ISO/IEC, "ISO/IEC 9075-1:2016, "SQL - Part 1: Framework"," 2016. [Online]. Available: https://www.iso.org/standard/63555.html

[22] ——, "ISO/IEC 9075-2:2016, "SQL - Part 2: Foundation"," 2016. [Online]. Available: https://www.iso.org/standard/63556.html

[23] G. B. Randolph, "The forest and the trees: Using Oracle and SQL Server together to teach ANSI-standard SQL," in *Proceedings of the 4th ACM Conference on Information Technology Curriculum (CITC)*, ser. CITC4 '03. New York, NY, USA: ACM, 2003, pp. 234–236. [Online]. Available: http://doi.acm.org/10.1145/947121.947174

[24] T. Taipalus and P. Perälä, "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)*, ser. SIGCSE '19. New York, NY, USA: ACM, February-March 2019, pp. 198–203.

[25] M. Axelsen, A. F. Borthick, and P. L. Bowen, "A model for and the effects of information request ambiguity on end-user query performance," *ICIS 2001 Proceedings*, p. 68, 2001. [Online]. Available: http://aisel.aisnet.org/icis2001/68

[26] A. Borthick, P. L. Bowen, D. R. Jones, and M. H. K. Tse, "The effects of information request ambiguity and construct incongruence on query development," *Decision Support Systems*, vol. 32, no. 1, pp. 3–25, November 2001. [Online]. Available: https://doi.org/10.1016%2Fs0167-9236%2801%2900097-5

[27] G. I. Casterella and L. Vijayasarathy, "Query Structure and Data Model Mapping Errors in Information Retrieval Tasks," *Journal of Information Systems Education*, vol. 30, no. 3, pp. 178–190, 2019. [Online]. Available: http://jise.org/Volume30/n3/JISEv30n3p178.pdf

[28] J. S. Davis, "Experimental investigation of the utility of data structure and E-R diagrams in database query," *International Journal of Man-Machine Studies*, vol. 32, no. 4, pp. 449 – 459, 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020737305801427

[29] R. L. Leitheiser and S. T. March, "The influence of database structure representation on database system learning and use," *Journal of Management Information Systems*, vol. 12, no. 4, pp. 187–213, March 1996. [Online]. Available: https://doi.org/10.1080%2F07421222.1996.11518106

[30] P. Reisner, "Use of psychological experimentation as an aid to development of a query language," *IEEE Transactions on Software Engineering*, vol. SE-3, no. 3, pp. 218–229, May 1977. [Online]. Available: https://doi.org/10.1109%2Ftse.1977.231131

[31] C. Welty and D. W. Stemple, "Human factors comparison of a procedural and a nonprocedural query language," *ACM Transactions on Database Systems*, vol. 6, no. 4, pp. 626–649, December 1981. [Online]. Available: https://doi.org/10.1145%2F319628.319656

[32] R. B. Buitendijk, "Logical errors in database SQL retrieval queries," *Computer Science in Economics and Management*, vol. 1, no. 2, pp. 79–96, 1988. [Online]. Available: https://doi.org/10.1007%2Fbf00427157

[33] P. Reisner, "Human factors studies of database query languages: A survey and assessment," *ACM Computing Surveys*, vol. 13, no. 1, pp. 13–31, March 1981. [Online]. Available: http://doi.acm.org/10.1145/356835.356837

[34] C. J. Date, "Critique of the SQL database language," *SIGMOD Record*, vol. 14, no. 3, November 1983.

[35] J. E. Hollingsworth, "Teaching query writing: An informed instruction approach," in *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, ser. ITiCSE '08. New York, NY, USA: ACM, 2008, pp. 351–351. [Online]. Available: http://doi.acm.org/10.1145/1384271.1384393

[36] R. Zilligen and A. Hidayat, "A misconception module to a database courseware," in *Proceedings of the 46th ACM Annual Southeast Regional Conference (ACM-SE)*, ser. ACM-SE 46. New York, NY, USA: ACM, 2008, pp. 529–530. [Online]. Available: http://doi.acm.org/10.1145/1593105.1593250

[37] R. Mason, C. Seton, and G. Cooper, "Applying cognitive load theory to the redesign of a conventional database systems course," *Computer Science Education*, vol. 26, no. 1, pp. 68–87, January 2016. [Online]. Available: https://doi.org/10.1080%2F08993408.2016.1160597

[38] T. Taipalus, "The effects of database complexity on SQL query formulation," *Journal of Systems and Software*, vol. 165, p. 110576, 2020.

[39] H. Al-Shuaily and K. Renaud, "SQL patterns - a new approach for teaching SQL," in *8th HEA Workshop on Teaching, Learning and Assessment of Databases (TLAD)*, 2010, pp. 29–40.

[40] L. Vijayasarathy and G. Casterella, "The effects of information request language and template usage on query formulation," *Journal of the Association for Information Systems*, vol. 17, no. 10, pp. 674–707, October 2016. [Online]. Available: https://doi.org/10.17705%2F1jais.00440

[41] T. Taipalus, "Teaching Tip: A Notation for Planning SQL Queries," *Journal of Information Systems Education*, vol. 30, no. 3, pp. 160–166, 2019. [Online]. Available: http://jise.org/Volume30/n3/JISEv30n3p160.pdf

[42] G. I. Casterella and L. Vijayasarathy, "An Experimental Investigation of Complexity in Database Query Formulation Tasks," *Journal of Information Systems Education*, vol. 24, no. 3, pp. 211–221, 2013. [Online]. Available: http://jise.org/Volume24/24-3/pdf/Vol24-3pg211.pdf

[43] G. Qian, "Teaching SQL: A divide-and-conquer method for writing queries," *Journal of Computing Sciences in Colleges*, vol. 33, no. 4, pp. 37–44, Apr. 2018. [Online]. Available: http://dl.acm.org/citation.cfm?id=3199572.3199577

[44] L. Sundin and Q. Cutts, "Is it feasible to teach query programming in three different languages in a single session?: A study on a pattern-oriented tutorial and cheat sheets," in *Proceedings of the 1st ACM UK & Ireland Computing Education Research Conference*, ser. UKICER. New York, NY, USA: ACM, 2019, pp. 7:1–7:7. [Online]. Available: http://doi.acm.org/10.1145/3351287.3351293