So many brackets! An analysis of how SQL learners (mis)manage complexity during query formulation

Daphne Miedema
d.e.miedema@tue.nl
Eindhoven University of Technology
Artificial Intelligence and Data Engineering (AIDE) Lab
Eindhoven, the Netherlands

George Fletcher
g.h.l.fletcher@tue.nl
Eindhoven University of Technology
Artificial Intelligence and Data Engineering (AIDE) Lab
Eindhoven, the Netherlands

Efthimia Aivaloglou
e.aivaloglou@liacs.leidenuniv.nl
Leiden Institute of Advanced Computer Science
Leiden, The Netherlands
Open Universiteit
Heerlen, The Netherlands

ABSTRACT

The Structured Query Language (SQL) is a widely taught database query language in computer science, data science, and software engineering programs. While highly expressive, SQL is challenging to learn for novices. Various research has explored the errors and mistakes that SQL users make. Specific attributes of SQL code, such as the number of tables and the degree of nesting, have been found to impact its understandability and maintainability. Furthermore, prior studies have shown that novices have significant issues using SQL correctly, due to factors such as expressive ease, existing knowledge and misconceptions, and the impact of cognitive load.

In this paper we identify another factor: self-inflicted query complexity, where users hinder their own problem solving process. We analyse 8K intermediate and final student attempts to six SQL exercises, approaching complexity from four perspective: correctness, execution order, edit distance and query intricacy. Through our analyses, we find that our students are hindered in their query formulation process by mismanaging complexity through writing overly elaborate queries containing unnecessary elements, overusing brackets and nesting, and incrementally building queries with persistent errors.

CCS CONCEPTS

• **Information systems → Structured Query Language**; • **Social and professional topics → Computing education**.

KEYWORDS

SQL, Databases, Complexity, Novice, Education

ACM Reference Format:
Daphne Miedema, George Fletcher, and Efthimia Aivaloglou. 2022. So many brackets! An analysis of how SQL learners (mis)manage complexity during query formulation. In *30th International Conference on Program Comprehension (ICPC '22), May 16–17, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3524610.3529158

ICPC '22, May 16–17, 2022, Virtual Event, USA

ACM ISBN 978-1-4503-9298-3/22/05.
https://doi.org/10.1145/3524610.3529158

1 INTRODUCTION

Education in database query languages is part of typical Computer Science, Software Engineering, and Data Science undergraduate curricula. Relational Databases are widely taught and used in practice. Their corresponding query language is the Structured Query Language (SQL). SQL is highly expressive, but with high expressiveness comes high complexity. As a result SQL is difficult to learn, as has been demonstrated by earlier studies examining errors made by learners [2, 4, 5, 26]. The most comprehensive investigation to date has been undertaken by Taipalus, Siponen and Vartiainen [37], who distinguish between syntactic, semantic, and logical errors and complications. Various researchers have examined reasons why novices make many mistakes [1, 6, 17, 19, 28]. One important factor is the students' existing knowledge, which influences their approach [28] and can lead to misconceptions [17].

The effects of program complexity on comprehension and cognitive load have also been widely studied [7]. Specifically for SQL, high complexity is problematic. As an illustration, see Listing 1 (and its corresponding question in Table 1), two subsequent queries written by a participant in our study. Specifically, the complexity of the second attempt makes the query close to incomprehensible.

```
SELECT pName, sName
FROM purchase
WHERE price = (SELECT MAX (PRICE)
FROM purchase) and pName = (SELECT pID
FROM purchase
WHERE price = MAX (price))

SELECT pName, sName
FROM purchase
WHERE price = MAX (price) and pName = (SELECT pName
FROM product
WHERE pID = (SELECT pID
FROM purchase
WHERE price = MAX (price)) and sName = (SELECT sName
FROM store
WHERE sID = (SELECT sID
FROM purchase
WHERE price = MAX (price)))
```

Listing 1: Participant 99, two subsequent attempts on exercise 5.

This high complexity has a large impact both cognitively and performance-wise. First, the cognitive load associated with SQL [4, 19, 21], as well as the relation between complexity and accuracy play a role. Siau et al. found an inverse relation between query complexity and accuracy [32] and Ion finds that complex SQL queries (defined through element count) are more prone to semantic errors [10]. Taipalus found that the complexity of the database structure leads to unnecessary complications and difficulties in refactoring queries [35]. Second, from a technical databases perspective, complexity is also the subject of much research. For example, progression in the capabilities of AI with regards to Natural Language processing has led to renewed interest on the topic of Natural Language interfaces for SQL, as AI has progressed to the point where it can now be used for queries that are more elaborate than plain SELECT FROM WHERE queries [30]. Maintainability and readability is also affected by specific complexity aspects of SQL queries, such as the number of tables and the the degree of nesting within the query [22].

In this paper we investigate the problem of complexity in SQL query formulation. The complexity of SQL does not just result in syntax and semantic errors, but can also hinder comprehension for novices during query formulation. Our research question is: *How do students deal with complexity during query formulation?* To analyze SQL complexity throughout the query formulation process, we approach complexity in our dataset from four angles: correctness, execution order, edit distance and query intricacy. We examine the intermediate and final queries that 104 students submitted to the database as solutions for six exercises, grouping by participant and exercise number. This series of attempts gives us insight into how students approach query building and how they manage and mismanage complexity during query formulation.

2 RELATED WORK

Confusion while coding can be caused by the cognitive processes involved in the retrieval of information from long-term memory, short-term memory, and working memory [8, p. 6]. All three of these processes play a role in the complexity of writing SQL queries. Regarding *long-term memory* requirements, SQL has strict syntax and low expressive ease, which is defined as "the syntactic flexibility permitted when formulating queries" [6, page 895]. During SQL query formulation, *short-term memory* needs to store and process information about the database schema [19, 21, 35] and the WHERE clause conditions [4]. Finally, the *working memory* and imposed cognitive load are affected by the lack of syntactic locality in SQL [1], which means that a small change in semantics (for example ∃ to ∀) requires a largely different query. This is aggravated by the fact that DBMS only responds to syntax errors, making it difficult for users to find semantic errors [19], as there is often no indication on how to locate these errors.

Another possible source of confusion for SQL novices is their prior knowledge. Before studying SQL, most novices have learned to program in imperative languages. This can be a hindrance, as SQL is a declarative language. Contrary to writing code in imperative languages that they have learned, in a declarative language novices cannot split up the problem in steps but have to apply set-based thinking [28]. Prior exposure to different programming languages has also been found to cause misconceptions [29], for example due to the differences in the notations and their semantics (for example, see the differences in usage of = and == for assignment and comparison). A recent qualitative think-aloud study on the causes of errors of SQL novices identified specific misconceptions stemming from prior course knowledge, along with generalization-based misconceptions, language-based misconceptions, and misconceptions due to an incomplete or incorrect mental model [16].

Finally, there are factors related to SQL syntax. It is strict and abstract, which leads to low expressive ease and errors [6]. This is compounded by typical Database Management Systems (DBMSs) only responding to syntax errors, leaving students and users to fend for themselves in the case of semantic errors [19].

The above-mentioned factors make SQL query formulation complex and error prone, especially for novices. In the past decades, several researchers have studied and categorized the errors made by SQL novices. Early work has discussed basic errors such as omissions and punctuation errors [26, 33]. More recent studies distinguish between syntax errors [2] and semantics errors [4], or discuss both [3, 23]. Brass and Goldberg distinguish between two types of semantically incorrect answers [5]: queries for which we need to know the question to see whether they are correct, and incorrect queries regardless of the question (e.g. due to a conflicting WHERE clause resulting in an empty answer). Taipalus, Siponen and Vartiainen also use these two classes: they call the former category *logical errors*, and the latter *semantic errors* [37].

Closely related to our work on SQL complexity is the work by Taipalus and Perälä, who define a category of problems called *complications* [36]. Complications are unnecessary elements that make it difficult to understand a query, although the query is correct. They examined the persistence of different categories of errors in students' attempts to solve queries, and found that complications and logical errors were the most persistent errors, whereas syntax and semantic errors were more likely to be fixed [36].

Also related to the study in this paper is the study by Migler and Dekhtyar, who also examine intermediate query attempts. They measure success rate, time spent per exercise, number of attempted solutions, and number of sessions per exercise, some of which overlap with our measures. They find that specific concepts, such as self joins, correlated subqueries and equal subqueries are difficult for students to master [18]. They argue that we should evaluate student learning by measures beyond success and errors.

Our work is related to and builds on the aforementioned studies by Taipalus et al. [36] and Migler et al. [18]. Similar to Migler and Dekhtyar [18], we analyze both intermediate and final attempts to SQL problems. In our work, however, we are focusing on complexity, and are therefore using a different set of metrics. By focusing on complexity, we can examine the causes underlying the high error prevalence in SQL formulation by novices. The other difference is that their study is more extensive than previous studies, providing the students with 116 distinct SQL exercises, versus studies with 15 [36] or 7 exercises [3]. In our study we have included 6 exercises that capture the basic concepts of SQL as mentioned by Migler and Dekhtyar: selections, joins, grouping, aggregation, and nesting or subqueries [18]. Taipalus and Perälä [36] lay another foundation for this work by looking at complexity from the perspective of persistent errors: errors that were not resolved by the student. This

Table 1: The exercises to be answered by the participants. Concepts with a ♢ are used by Ahadi et al. [3] and Taipalus et al. [37], concepts with a ♠ were introduced by Taipalus et al. [37].

	Exercise text	SQL concepts
Exercise 1	List all the product ids that were bought by at most two different customers.	self-join ♢, does not exist ♠
Exercise 2	List the customers (id and name) who purchased on the same date both a product with name 'Onions' and a product with name 'Coffee'.	self-join ♢, does not exist ♠
Exercise 3	List the ids of the customers that made a purchase at every store.	correlated subquery ♢, does not exist ♠
Exercise 4	Find the names of the store-chains that on average sell products in quantities of more than 4.	group by with having ♢, aggregate functions ♠
Exercise 5	Find the name of the product that is sold for the highest price (ever) and the name of the store that sold that item for that price.	aggregate functions ♠
Exercise 6	Find the largest difference in price for a product in stock (i.e., in the inventory of a store) on '2018-08-23' between two different stores.	aggregate functions ♠

inability to solve problems reflects a problem in SQL complexity. Our work builds on this by quantifying complexity in the plain text queries through four different measures: correctness, execution order, edit distance and query intricacy.

3 METHOD

For our exploratory analysis, we recruited students from the authors' institution's introductory Databases course. This course is taught in the second year of the Computer Science Bachelor. Beforehand, all participants had been introduced to SQL in a mandatory first-year course on Data Analytics. In total, 104 out of 450 students participated in our study. Note that the participant numbers exceed 104 as we numbered each student in the course.

Ethics. Gathering of the data has been approved by the Ethical Review Board of the authors' institution.

3.1 Data Collection and Filtering

We provided all participating students with a Jupyter notebook [14] for a graded assignment on SQL. The notebook contained cells that enable the students to run Python code. The assignment consisted of six exercises testing different concepts (Table 1) to be evaluated over the schema in Table 3. A correct query is defined as one that has the same result table as any of the predefined correct queries introduced by the teacher. Multiple answers may be correct, all correct answers per exercise can be found in our supplementary material[1]. The expected answers contain the SQL concepts as indicated in Table 1.

Students were free to attempt the exercises in any order they preferred, including switches between exercises, as the questions did not build on each other. As we aimed to examine the query formulation process throughout all attempts without external influences, the students were not given feedback on correctness, nor could they compare their answer to a correct result table.

The notebook was pre-filled with the six exercises, plus corresponding code cells that allowed the students to write and execute SQL queries immediately. The code cells were named to determine which exercise a student was trying to solve. The notebook also included logging functionality, such that the contents of each executed Python code cell were written to a file. From this, we could then extract all attempts that the students had made for each exercise. In total, we captured 10,818 attempts over six exercises.

Initial data cleaning concerned code cell names. Some students had created extra code cells in the notebooks, in which they wrote intermediate queries or drafted items. These had different names than the predefined cells. As we could not always determine the corresponding exercise, we excluded all such code cells. We also removed all queries that were not strings, and those that contained the query placeholder. This reduced the total number of attempts from 10,818 to 8,829.

3.2 Data processing

To answer our research question we explore query complexity from four distinct angles: correctness and attempts, execution order, edit distance, and query complexity. These four perspectives provide a complimentary analysis of how students manage complexity. To learn more about our data and to replicate our analysis, we refer the reader to our supplementary material[1].

Correctness and attempts. Analysis of correctness and attempts gives us a direct insight into the difficulty of the exercises for students. The first step for this analysis was to calculate correctness for all queries in our collection. We used four categories: correct, semantic error, syntax error, and time-out. We ran each query and compared the result table against the result tables of all instructor-supplied correct answers. Correct answers were those where the result table of the attempt matched exactly, syntax errors and time-outs (5 seconds) were determined by the DBMS, and all other answers were categorized as semantic errors.

Execution order. Execution order plots are a visualization to analyze the order in which students attempted each exercise, and *when* in their problem solving process they returned to which exercise. This provides us insights into how students dealt with errors.

For the execution order plots, no preprocessing of the data was required. We generated plots showing the order in which exercises were attempted, and whether each attempt was correct or not. One such plot was generated per participant, from which we gained insight into their query formulation process. Guided by these plots, we examined the query logs of various participants to gather more insight into the query formulation process.

[1] https://doi.org/10.6084/m9.figshare.19430375

Table 2: Correctness and attempts for all exercises.

	total attempts	correct	syntax error	semantic error	timeout	% correct attempts	% correct participants	attempted by
1	1,332	284	333	678	37	21.32	90.43	94
2	1,771	158	509	1,087	17	8.92	81.82	88
3	1,489	220	560	691	18	14.78	92.13	89
4	1,671	169	453	1,030	19	10.11	73.03	89
5	1,239	144	455	639	1	11.62	73.86	88
6	1,327	3	426	894	4	0.23	1.11	89

Table 3: The database schema. Underlined attributes indicate the primary keys.

Table name	Attributes
customer	cID, cName, street, city
store	sID, sName, street, city
product	pID, pName, suffix
shoppinglist	cID, pID, quantity, date
purchase	tID, cID, sID, pID, date, quantity, price
inventory	sID, pID, date, quantity, unit-price

Edit distance. Edit distance analysis shows how different each newly defined query is from the one before. Do students keep struggling on an exercise they don't manage to answer, or (at what point) do they start over?

We decided to use an adapted version of Levenshtein distance in which we count the differences in *words* instead of the differences in *characters*. SQL queries are different from plain text in the sense that there are a lot of reserved keywords that cannot be spelled differently. Additionally, almost all other elements in the query are static too: table and column names only have one correct spelling. A misspelling, regardless of its size, will result in a syntax error. Combining these factors, we chose to measure word difference over character difference.

Before analyzing the data, we excluded all attempts that had an edit distance of 0. The repeated execution of the exact same query was common in our data, for a total of 1,114 repeated attempts (12.5 percent). This leaves a total of 7,757 attempts for analysis. Then, we calculated the *Change Ratio*, which is defined as (1 - Levenshtein Distance Ratio), again using the adapted definition of Levenshtein.

Based on the statistics found, we again examined some query logs for further insights.

Query intricacy. Finally, we analyzed the intricacy of all attempts by calculating three metrics on the query text. The scores show the complexity of queries written by the students in relation to those by the teachers. First of all, we counted the number of query elements: we split each query into words and other elements. This measure of intricacy can represent any type of text, not just queries. Next, we considered SQL from a Domain-Specific Language perspective. As a second measure we count the sets of brackets per attempt, as these can be used for complex elements such as subqueries and aggregations, as well as being used for clarifying more complex queries. To approximate the number of subqueries in each attempt, we count the number of SELECT clauses as the third metric. This is the same as the NN-measure by Piattini and Martinez [22].

4 RESULTS

4.1 Correctness and attempts

Our analysis on overall correctness and attempts shows that on average, some exercises were more difficult than others. For the full statistics, see Table 2. Regarding the number of attempts, exercises 2 and 4 have higher numbers than exercises 1 and 3, but lower numbers for correct attempts. Although exercise 3 has a high number of correct attempts, it also has the highest number of syntax errors. Clearly, students struggled writing the correct syntax for this question. Exercise 6 is an extreme outlier regarding correctness, with the three correct answers all coming from only one participant.

4.2 Execution order

Our analysis on execution order shows that students have different ways of attempting the exercises. To illustrate this, we show the execution order plots for participants 31 (Figure 1) and participant 99 (Figure 2). Some work in a linear order, such as participant 31, whereas others approach the work in a more random order, such as participant 99. Some attempt each exercise more than ten times, whereas others make only one or two attempts before switching to a different exercise. Execution order and switching can be attributed partially to perceived exercise complexity: when an exercise seems difficult, students might want to skip it and think about it while answering another exercise in the meantime.

The execution order plots show many repeated attempts that resulted in syntax or semantic errors, such as attempts 44 through 50 for participant 99, and attempts 82 through 88 for participant 31. When we inspect the raw query texts, we see evidence for self-inflicted complexity: students move from simple queries that produce errors, to more complex queries that still produce errors. For participant 99 we can see such behavior in the series of seven syntax errors for Exercise 5. This student tries to get closer to the syntactically correct answer, before solving their syntax errors. They extend their query with extra SELECT clauses and brackets multiple times (see Listing 1), before taking a step back and simplifying again. The second attempt specifically lacks indentation to represent how the subqueries relate and is therefore difficult to understand. It seems highly likely that the student lacked knowledge to correctly answer this exercise, especially since the first attempt is already close to the correct answer.

The aforementioned process of adding complexity before solving syntax errors is counter-productive, and in this case might have even lead to the participant not solving the problem. Furthermore, the student does not use indentation at all, which makes it more difficult for them to comprehend which subquery returns what.

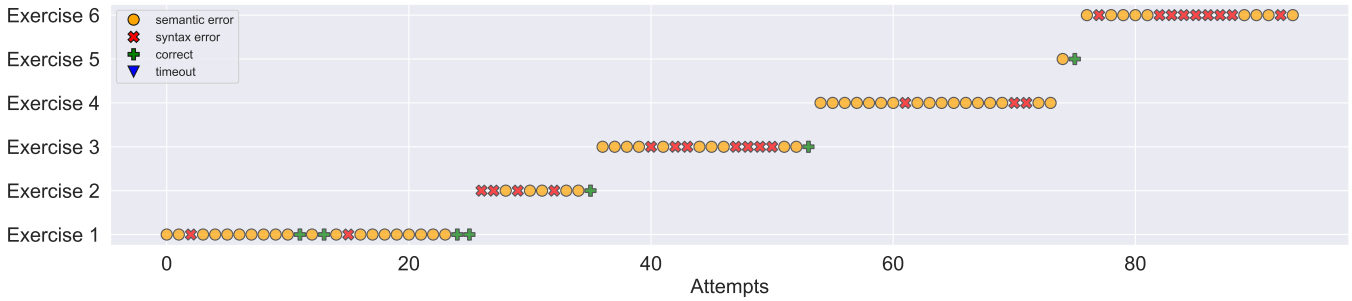

Figure 1: An execution order plot for participant 31. This participant approaches the problems in order.

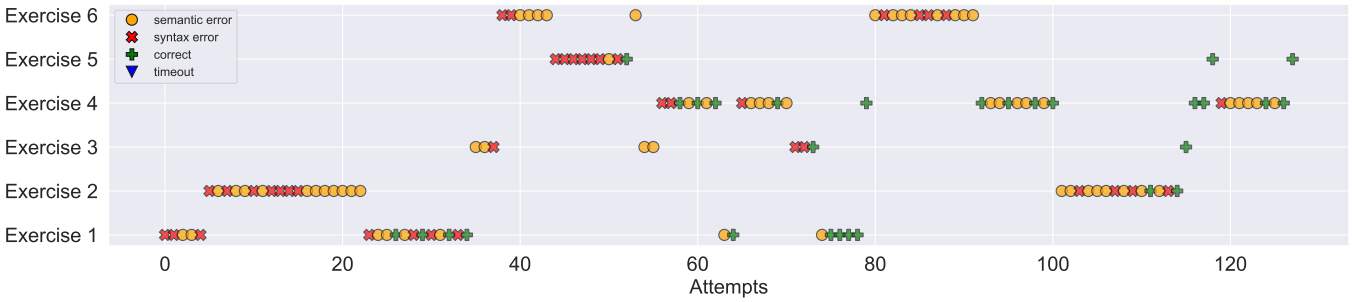

Figure 2: An execution order plot for participant 99. This participant switches between exercises often.

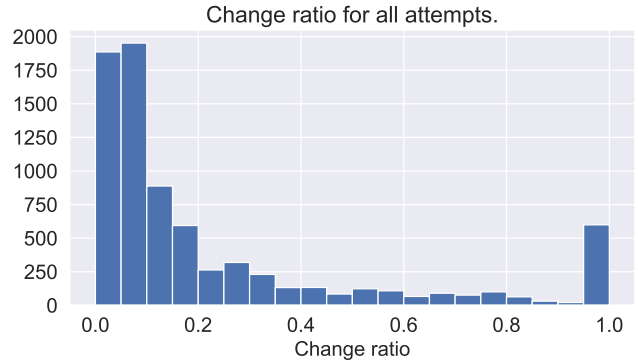

Figure 3: Change ratio < 0.1 is most common, meaning that participant typically make small changes.

4.3 Edit distance

Our analysis shows that during query formulation, students stick with their initial attempts and hardly ever start over. After excluding all attempts with edit distance 0, we are left with 7,757 attempts. In Figure 3 we see that the majority of attempts (49.5 percent or 3837 attempts) has a change ratio less than 0.1.

Besides these aggregated change ratios, we also examined our students' query formulation process by plotting the absolute edit distance per participant per exercise. A high edit distance can mean that a student is changing from one approach to another. For example, they change from an approach utilizing a subquery to one that does not have a subquery, or they remove a large part from their query to start over. We should keep in mind that a large edit distance may mean both a large addition to the query, or a large removal. For examples, see the queries of participant 15 in Appendix A. Once we examine the contents of the query logs in combination with these plots, we find some interesting behaviour.

First of all, let us consider participant 15 in Figure 4. The values on the y-axis reflect the difference in placement and content of the words within the query. In the plot we circled some queries with similar edit distances. These occurrences regularly represent a switch between two solutions: the edit distance between two queries is the same, no matter which direction you switch in. Suppose a students is working on query formulation using approach a. Then, they start on approach b, which has an edit distance of x from approach a. Then, once they return from approach b back to approach a, this again shows an edit distance of x. For a concrete example we again refer to the queries in Appendix A. So, a set of similar edit distances reflects a temporary change of approach.

The edit distance plot for participant 381 (Figure 5) shows gradually increasing edit distances between attempt 20 and 35. Inspection of the query logs shows that this participant tried to solve the problem in an unconventional way: manually checking the purchase IDs that should be included, and then hard-coding them in a query. For an excerpt from the process, see Listing 2. These switches result in the slightly increasing edit distance, which is a bit misleading as there is no real progress happening.

To reiterate, from our plots of edit distance and change ratio we can conclude that our participants hardly start over. Even if the edit distance indicates a restart, this often represents only a quick check of the basic query. This behavior might be a result of sunk cost fallacy.

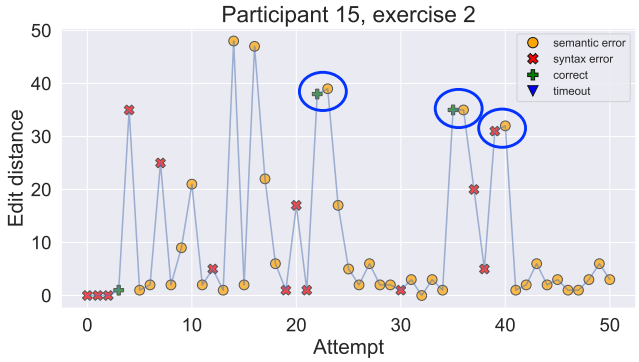

Figure 4: Participant 15 makes large changes throughout their attempts with differences of up to 39 words.

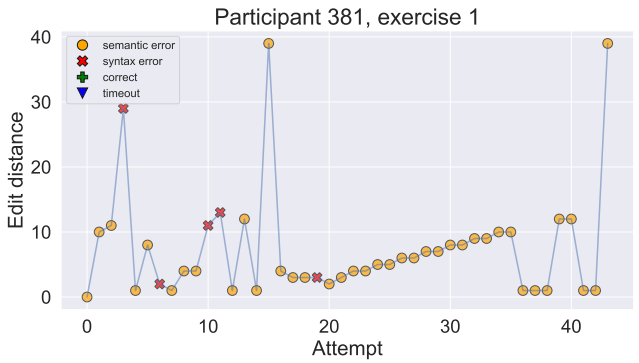

Figure 5: Participant 381 makes edits in a gradual manner.

4.4 Query intricacy

Finally, our analysis on query intricacy highlights extreme values for the measures introduced in subsection 3.2. The scores for query elements can be found in Table 4 and scores on brackets in Table 5. The statistics on element count show that the average attempt in the data set contains 42.7 elements. We assume that students work towards a correct answer and thus expect the majority of queries to have a lower complexity than the correct answer. However, we find many queries with higher complexity, as can be seen in the right-hand column of the table. We even find 230 attempts counting over 100 elements (2.6%).

Our analysis on bracket sets in Table 5 shows that students use up to ten sets of brackets, that are nested up to five levels deep. The exercises in this study require a maximum of two sets of brackets and no nesting. Although students might use extra brackets for clarification, we deem queries with five or more sets of brackets too convoluted.

When we examine the queries corresponding to these attempts with high intricacy scores, we find overly complex SQL text as a result of students' continuous struggles. For an example, see Listing 3, in which a student wrote an attempt of 124 elements and with eight sets of brackets.

Brackets do not always indicate subqueries. They are also used for view definition (Listing 3) and aggregation. To reflect on the number of subqueries, we therefore also counted the number of SELECT clauses per query. We found that queries used between zero and eight SELECT clauses, with 160 queries containing four or more SELECT clauses. Given that the exercises required only one or two SELECT clauses, this is overly complex. Also, the presence of queries without SELECT clauses is interesting, as all exercises required access to the database through SQL.

```
18  SELECT * FROM purchase
19  SELECT * FROM purchase NOT IN (10)
20  SELECT * FROM purchase WHERE pID NOT IN (10)
21  SELECT * FROM purchase WHERE pID = 14
22  SELECT * FROM purchase WHERE pID NOT IN (10, 14)
23  SELECT * FROM purchase WHERE pID = 16
24  SELECT * FROM purchase WHERE pID NOT IN (10, 14, 16)
25  SELECT * FROM purchase WHERE pID = 9
26  SELECT * FROM purchase WHERE pID NOT IN (10, 14, 16, 9)
```

Listing 2: Participant 381, exercise 1, 9 consecutive attempts at solving the problem in a round-about way.

```
select distinct s.sName
from store as s
where s.sName in (
    with store_chains(sName, number) as (
        select s.sName, count(distinct s.sID)
        from store as s
        group by s.sName)
    select sc.sName
    from store_chains as sc
    where sc.number>1)
and s.sName in(
    with store_chains_qua(sName, quantity) as (
        select s.sName, avg(p.quantity)
        from purchase as p, store as s
        where p.sID=s.sID group by s.sName)
    select sa.sName
    from store_chains_qua as sa
    where sa.quantity>4)
```

Listing 3: A highly intricate attempt by participant 410, demonstrating self-inflicted complexity. Indentation and newlines added for readability.

The aforementioned high numbers indicate a high level of query intricacy, but do not reveal anything about the development of complexity. To quantify this further, we examined the student behavior after a syntax error. Syntax errors bring a hard stop to query execution, as the system will not return any results. In such cases, resolving the syntax error would often be achieved through simplification. However, we found that it was a common behavior to increase intricacy after a syntax error. 36.8% of erroneous attempts were followed by a subsequent attempt with a higher number of query elements, with participants adding three or more elements in 17.1% of cases. 8.6% of the erroneous attempts were followed by a subsequent attempt with a higher number of bracket sets.

Overall, inspection of the query logs shows that students keep adding more and more elements to their queries to try to solve their problems (self-inflicted complexity). In these cases starting over would likely be a more efficient approach.

Table 4: Number of query elements.

	mean	stdev	max	complexity of correct answer	#attempts of higher complexity
Exercise 1	34.0	24.7	122	66.3	145
Exercise 2	56.3	28.7	187	75.5	464
Exercise 3	32.7	17.1	174	45	289
Exercise 4	41.0	20.7	124	54	419
Exercise 5	40.9	18.1	120	51.5	267
Exercise 6	48.2	24.6	140	49	571
Overall	42.7	24.4	187	-	-

5 DISCUSSION

SQL query formulation is complex and error prone for novices. Most attempts in our dataset were found to contain either syntax or semantic errors. This was expected, as research has shown errors to be common [2, 3, 5, 18, 23, 26, 33]. Towards understanding the reasons for these errors, our analysis of intermediate attempts gives insight in novices' query formulation process. Our findings indicate that students are hindered in their query formulation process by mismanaging complexity, mainly through building overly elaborate queries containing unnecessary elements. The analysis also provided us with concrete examples of ways in which query complexity is increased: many brackets, many subqueries, many query elements, and lack of indentation to assist in query comprehension.

Evidence of mismanaging complexity was conveyed by each of the four angles from which we approached the analysis. The high-level analysis of *correctness and attempts* indicated that most attempts were erroneous, indicating struggles to answer the exercise correctly. Analyzing the *edit distance*, we found that students tend to stick with their initial attempts and hardly ever start over, mostly adding elements to their initial queries. This enrichment of the queries was found to go beyond the required query complexity level; the analysis of *query intricacy* indicates an unnecessarily large number of brackets and nesting, and even queries of extremely high complexity. This finding is in line with existing work reporting that complication errors are common and persistent [36]. Our analysis also highlighted that the enrichment of queries does not usually lead students to the correct result, since the *execution order* analysis indicated a large number of subsequent erroneous attempts per query. During query formulation, the students often attempted to come closer to the correct answer by editing simple queries that produce errors to more complex queries that still produce errors.

On top of complexity at a query level, the analysis of the execution order indicates self-inflicted complexity at the assignment level. Instead of following a linear approach to answering the exercises, students would often switch after a few attempts per exercise. Switching between exercises could be advantageous once another exercise gives you some insights, but could also be associated with an increase in cognitive load or working memory requirements.

Prior work in the programming education research community has mostly focused on the effects of the complexity of code written in imperative languages. Due to the declarative nature of SQL, comparison with our findings is not straightforward as imperative metrics are not necessarily compatible. Such metrics, which have been used in prior work, include plan depth and plan interactivity [7], McCabe's Cyclomatic Complexity [13, 20] and Halstead's metric [20]. Although the applied metrics are different, parallels can be drawn: several findings from prior work on programming education might apply to SQL. Program decomposition is an issue for students, and complexity metrics can reflect their progress [13] -the problem decomposition issue becomes apparent in our work on SQL complexity too. Algorithmic complexity was identified as a problem for novice programmers [27], which expert programmers are able to tackle by adapting effective strategies. For SQL, those strategies might relate to the use of query templates [24].

Compared to the work by Yang et al. [38] on Levenshtein distance of SQL queries, our findings differ. The difference between our approach and theirs is that they use standard Levenshtein, and only calculate the distance between each attempt and the final attempt. They find a reflection of trial-and-error approach with oftentimes small edit distances (mostly for syntax errors), and a reflection of divide-and-conquer with larger edit distances for semantic errors [38]. This does not match our findings. This may be in part due to the fact that our students did not receive feedback during query formulation on whether their attempt was correct, except for any syntax errors. Thus, students did not know when their attempt had a semantic error. However, we also saw different behavior for syntax errors in our population: the focus was not on repairing syntax, but on answering the exercise. This mismanagement of complexity was typically counterproductive for the student.

Our findings also relate to the work by Taipalus [35], who reports that the complexity of a database schema influences correctness in query formulation. One important cause, according to him, is exceeding working memory capacity: a larger schema implies more elements to remember. This could extend to queries, with a larger query naturally meaning a larger load on working memory. Unexpectedly, in our work we find that students do not simplify their queries in case of errors; instead, they regularly make their query even more complex, without solving the syntax error.

5.1 Implications for teaching practice

Our analyses revealed that students mismanage query complexity. We believe that the underlying cause for this could be a fragmented understanding of SQL itself, as well as insufficient knowledge or incorrect perspective of how to decompose the query building process. Indeed, a similar problem has also been shown to occur in programming education: it has been found that students often encounter difficulties in understanding the task and decomposing the problem [25]. Decomposition can be supported by tracing programs: running a mental model that encompasses both the notional machine and the traced program [34]. As SQL is a declarative language, query execution can not be traced in the same way as can be done for programs written in imperative languages. This hinders the development of correct mental models and might support their 'black box' view of the DBMS. To assist in decomposition for SQL queries, solutions have been proposed for visualization of intermediate query results (eSQL [12] and SQLVis [17]), as well as concept maps of the evolution process of intermediate results of SQL statements [31].

Table 5: Attempts per bracket set count per question.

	Number of sets of brackets											Nesting depth					
	0	1	2	3	4	5	6	7	8	9	10	0	1	2	3	4	5
Exercise 1	692	401	175	39	22	3	0	0	0	0	0	692	467	139	4	0	0
Exercise 2	958	349	207	137	92	9	9	5	3	2	0	959	572	169	65	6	0
Exercise 3	298	413	293	252	140	57	32	4	0	0	0	298	650	512	24	5	0
Exercise 4	251	649	390	255	61	34	24	2	5	0	0	251	1013	360	38	9	0
Exercise 5	136	454	383	194	39	17	12	1	0	0	3	136	507	529	47	19	1
Exercise 6	299	281	262	230	125	53	44	16	16	0	1	300	556	430	39	2	0

We identify several approaches that can be applied in teaching practice towards educating students about the query formulation process and about managing its complexity. From cognitive science we know that a process that helps in dealing with complex problems is chunking, where information is combined and organized together into conceptually related groups, or chunks. Chunking has already been researched in programming teaching practice [15] by applying pattern-oriented instruction [11]. In SQL instruction, recent work in this direction has proposed the application of templates to divide-and-conquer query formulation problems [24].

Towards helping novices internalize SQL syntax, patterns and templates, two techniques that could help are retrieval practice and active elaboration [9, page 40]. Retrieval practice is based on the idea that learning is boosted when you try to retrieve data from memory, such as by learning with flashcards. Active elaboration is a method of reflecting on the topic to learn, and linking it to things you already know, such that the information fits better in long term memory. Indeed, the efficacy of practicing retrieval through spaced repetition has been demonstrated in programming education [39]. However, the application of those techniques to SQL instruction remains to be investigated.

5.2 Threats to validity

In the study we theorize that students suffer from self-inflicted complexity in the formulation of SQL queries. We substantiate this by calculating metrics and showcasing raw queries. As such, our study has appropriate descriptive validity and theoretical validity. Two factors that may have led to low generalizability (both externally and internally) are our population and the number and design of the exercises.

Our participant population and number of queries was sufficiently large. However, the fact that the analysis was done on participants from a single university, after taking one specific course, is a limitation. Also, prior experience that participants may have with SQL and programming, for which we did not capture information, might have affected the results.

Furthermore, we analyzed attempts to only six exercises, versus the larger sets in other papers. Although we cover many of the concepts that students find most difficult [36], we did not include the lowest level of simplicity: a query on a single table. It would be interesting to see whether the concept of self-inflicted complexity would also arise in such exercises. The design of the exercises, including the database schema and the wording, has also influenced the participants' attempts. While the indications for the self-inflicted complexity are externally valid, the factors mentioned above limit the generalizability on the exact answers and attempts.

In the design of the study we chose an approach with high ecological validity: the behavior of students within the study was highly similar to the behavior of students outside the study. We wanted to analyze the approach of the participants to the homework (execution order) as one of our four dimensions of complexity. In order to capture execution order, we enabled the participants to switch between exercises. Even though the exercises did not relate to each other, this might have affected the results in the other three dimensions of complexity because of the context switching that the students were enabled to do.

Finally, our students may have answered questions correctly without realizing it, because they did not receive feedback on correctness. As a result, the success rates reported in in Table 2 may look worse than they would have been in case of feedback on correctness. To mitigate this, we also report on the percentage of participants with at least one correct answer.

6 CONCLUSION

Existing research has shown that writing queries in SQL is complex, especially for novices. Students struggle with this complexity as they make many mistakes during query formulation. We have shown that students mismanage complexity in SQL by trying to persevere through syntactic and semantic errors, leading them to overly complex answers. We urge teachers to integrate SQL problem decomposition in their lectures.

The results of this quantitative study open up several directions for future work. First, a qualitative study can provide complementary insight into the problem solving process that the novices follow while building SQL queries and tackle errors. Furthermore, specific interventions could be researched for helping novices decompose SQL problems and manage complexity, such as teaching SQL patterns and templates. This can be guided by referencing the literature on program decomposition, such as the work by Keen and Mammen [13]. These interventions are not only applicable to the context of students, but also to practitioners aiming to learn or refresh their SQL knowledge. However, in this case, extra attention must be paid to the role of different background knowledge and experiences of practitioners in designing educational materials and methods. Finally, methods can be investigated for giving novices incentives to not write highly complex queries, e.g., by reporting their complexity scores compared to a correct answer.

REFERENCES

[1] Azza Abouzied, Joseph M Hellerstein, and Avi Silberschatz. 2012. Playful Query Specification with DataPlay. In *Proceedings of the VLDB Endowment*. 1938–1941.

[2] Alireza Ahadi, Vahid Behbood, Arto Vihavainen, Julia Prior, and Raymond Lister. 2016. Students' Syntactic Mistakes in Writing Seven Different Types of SQL Queries and its Application to Predicting Students' Success. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE)*. 401–406.

[3] Alireza Ahadi, Julia Prior, Vahid Behbood, and Raymond Lister. 2015. A Quantitative Study of the Relative Difficulty for Novices of Writing Seven Different Types of SQL Queries. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. 201–206.

[4] Alireza Ahadi, Julia Prior, Vahid Behbood, and Raymond Lister. 2016. Students semantic mistakes in writing seven different types of SQL queries. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. 272–277. https://doi.org/10.1145/2899415.2899464

[5] Stefan Brass and Christian Goldberg. 2006. Semantic errors in SQL queries: A quite complete list. *Journal of Systems and Software* 79, 5 (2006), 630–644.

[6] Hock Chuan Chan, Bernard C. Y. Tan, and Kwok Kee Wei. 1999. Three important determinants of user performance for database retrieval. *International Journal of Human-Computer Studies* 51, 5 (nov 1999), 895–918. https://doi.org/10.1006/IJHC.1999.0272

[7] Rodrigo Duran, Juha Sorva, and Sofia Leite. 2018. Towards an Analysis of Program Complexity From a Cognitive Perspective. In *Proceedings of the 14th ACM Conference on International Computing Education Research (ICER)*. 21–30. https://doi.org/10.1145/3230977.3230986

[8] Felienne Hermans. 2021. Decoding your confusion while coding. In *The Programmer's Brain*. Manning Publications, Chapter 1.

[9] Felienne Hermans. 2021. How to learn programming syntax quickly. In *The Programmer's Brain*. Manning Publications, Chapter 3.

[10] Claudiu Ion. 2021. *A Static-Based Approach to Detect SQL Semantic Bugs*. Master's thesis. Delft University of Technology.

[11] Vighnesh Iyer and Craig Zilles. 2021. Pattern Census: A Characterization of Pattern Usage in Early Programming Courses. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE)*. 45–51. https://doi.org/10.1145/3408877.3432442

[12] R. Kearns, S. Shead, and A. Fekete. 1997. A teaching system for SQL. In *Proceedings of the 2nd Australasian conference on Computer Science Education (ACSE)*. 224–231. https://doi.org/10.1145/299359.299391

[13] Aaron Keen and Kurt Mammen. 2015. Program decomposition and complexity in CS1. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE)*. 48–53.

[14] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. 2016. Jupyter Notebooks - a publishing format for reproducible computational workflows.. In *Proceedings of the 20th International Conference on Electronic Publishing (Elpub)*. https://doi.org/10.3233/978-1-61499-649-1-87

[15] Douglas Kranch. 2012. Teaching the novice programmer: A study of instructional sequences and perception. *Education and Information Technologies* 17 (09 2012), 291–313. https://doi.org/10.1007/s10639-011-9158-8

[16] Daphne Miedema, Efthimia Aivaloglou, and George Fletcher. 2021. Identifying SQL Misconceptions of Novices: Findings from a Think-Aloud Study. In *Proceedings of the 17th ACM Conference on International Computing Education Research (ICER)*. https://doi.org/10.1145/3446871.3469759

[17] Daphne Miedema and George Fletcher. 2021. SQLVis: Visual Query Representations for Supporting SQL Learners. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*.

[18] Andrew Migler and Alex Dekhtyar. 2020. Mapping the SQL learning process in introductory database courses. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE)*. 619–625. https://doi.org/10.1145/3328778.3366869

[19] Antonija Mitrovic. 1998. Learning SQL with a computerized tutor. In *Proceedings of the 29th ACM Technical Symposium on Computer Science Education (SIGCSE)*. 307–311.

[20] Jesús Moreno-León, Gregorio Robles, and Marcos Román-González. 2016. Comparing computational thinking development assessment scores with software complexity metrics. In *2016 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 1040–1045.

[21] William C. Ogden and Susan R. Brooks. 1983. Query languages for the casual user. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI)*. ACM Press, New York, New York, USA, 161–165. https://doi.org/10.1145/800045.801602

[22] Mario Piattini and Antonio Martinez. 2000. Measuring for Database Programs Maintainability. In *Proceedings of the 11th International Conference on Database and Expert Systems Applications (DEXA)*, Mohamed Ibrahim, Josef Küng, and Norman Revell (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 65–78.

[23] Kai Presler-Marshall, Sarah Heckman, and Kathryn T Stolee. 2021. SQLRepair: Identifying and Repairing Mistakes in Student-Authored SQL Queries. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, 199–210.

[24] Gang Qian. 2018. Teaching SQL: A Divide-and-Conquer Method for Writing Queries. *Journal of Computing Sciences in Colleges* 33, 4 (April 2018), 37–44.

[25] Yizhou Qian and James Lehman. 2017. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Transactions on Computing Education* 18, 1, Article 1 (Oct. 2017), 24 pages. https://doi.org/10.1145/3077618

[26] Phyllis Reisner. 1977. Use of Psychological Experimentation as an Aid to Development of a Query Language. *IEEE Transactions on Software Engineering* 3 (1977), 218–229.

[27] Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and teaching programming: A review and discussion. *Computer Science Education* 13, 2 (2003), 137–172.

[28] Shazia Sadiq, Maria Orlowska, Wasim Sadiq, and Joe Lin. 2004. SQLator - An online SQL learning workbench. In *Proceedings of the 9th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. 223–227. https://doi.org/10.1145/1026487.1008055

[29] Z. Scherz, D. Goldberg, and Z. Fund. 1990. Cognitive Implications of Learning Prolog—Mistakes and Misconceptions. *Journal of Educational Computing Research* 6, 1 (1990), 89–110. https://doi.org/10.2190/UHFF-4LNQ-63VA-Q60C

[30] Jaydeep Sen, Chuan Lei, Abdul Quamar, Fatma Özcan, Vasilis Efthymiou, Ayushi Dalmia, Greg Stager, Ashish Mittal, Diptikalyan Saha, and Karthik Sankaranarayanan. 2020. Athena++ natural language querying for complex nested sql queries. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2747–2759.

[31] S. Shin. 2020. Structured Query Language Learning: Concept Map-Based Instruction Based on Cognitive Load Theory. *IEEE Access* 8 (2020), 100095–100110. https://doi.org/10.1109/ACCESS.2020.2997934

[32] Ken Leng Siau, Hock Chuan Chan, and Kwok Kee Wei. 2004. Effects of Query Complexity and Learning on Novice User Query Performance With Conceptual and Logical Database Interfaces. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 34, 2 (mar 2004), 276–281. https://doi.org/10.1109/TSMCA.2003.820581

[33] John B. Smelcer. 1995. User errors in database query composition. *International Journal of Human-Computer Studies* 42, 4 (1995), 353–381.

[34] Juha Sorva. 2012. *Visual Program Simulation in Introductory Programming Education*. Ph.D. Dissertation. Aalto University, Espoo, Finland.

[35] Toni Taipalus. 2020. The effects of database complexity on SQL query formulation. *Journal of Systems and Software* 165 (2020), 110576. https://doi.org/10.1016/j.jss.2020.110576

[36] Toni Taipalus and Piia Perälä. 2019. What to expect and what to focus on in SQL query teaching. In *In Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE)*. 198–203. https://doi.org/10.1145/3287324.3287359

[37] Toni Taipalus, Mikko Siponen, and Tero Vartiainen. 2018. Errors and Complications in SQL Query Formulation. *ACM Transactions on Computing Education* 18, 3 (2018).

[38] Sophia Yang, Ziyuan Wei, Geoffrey L Herman, and Abdussalam Alawini. 2021. Analyzing Patterns in Student SQL Solutions via Levenshtein Edit Distance. In *Proceedings of the 8th ACM Conference on Learning@Scale (L@S)*. 323–326.

[39] Iman YeckehZaare, Paul Resnick, and Barbara Ericson. 2019. A Spaced, Interleaved Retrieval Practice Tool That is Motivating and Effective. In *Proceedings of the 15th ACM Conference on International Computing Education Research (ICER)*. 71–79. https://doi.org/10.1145/3291279.3339411

A EXAMPLE OF REPEATED HIGH EDIT DISTANCE.

Attempt	Query	Correctness	Edit distance

12	`SELECT c.cID, c.cName`
	`FROM customer as c, purchase as p, product as pr`
	`WHERE p.cID = c.cID AND p.pID=pr.pID AND pr.pName = 'Coffee'`
	`AND (c.cID) IN (`
	`  SELECT c.cID`
	`  FROM purchase as p, product as pr`
	`  WHERE p.pID=pr.pID AND pr.pName = 'Onions' )`
	`AND p.date IN (`
	`  SELECT p.date`
	`  FROM purchase as p, product as pr`
	`  WHERE p.pID=pr.pID AND pr.pName = 'Onions' )`
	`GROUP BY SELECT c.cID, c.cName`
		syntax error	5

13	`SELECT c.cID, c.cName`
	`FROM customer as c, purchase as p, product as pr`
	`WHERE p.cID = c.cID AND p.pID=pr.pID AND pr.pName = 'Coffee'`
	`AND (c.cID) IN (`
	`  SELECT c.cID`
	`  FROM purchase as p, product as pr`
	`  WHERE p.pID=pr.pID AND pr.pName = 'Onions' )`
	`AND p.date IN (`
	`  SELECT p.date`
	`  FROM purchase as p, product as pr`
	`  WHERE p.pID=pr.pID AND pr.pName = 'Onions' )`
	`- GROUP BY SELECT c.cID, c.cName`
	`+ GROUP BY c.cID, c.cName`
		semantic error	1

14	`- SELECT c.cID, c.cName`
	`+ SELECT *`
	`FROM customer as c, purchase as p, product as pr`
	`- WHERE p.cID = c.cID AND p.pID=pr.pID AND pr.pName = 'Coffee'`
	`+ WHERE p.cID = c.cID AND p.pID=pr.pID`
	`- AND (c.cID) IN (`
	`-  SELECT c.cID`
	`-  FROM purchase as p, product as pr`
	`-  WHERE p.pID=pr.pID AND pr.pName = 'Onions' )`
	`- AND p.date IN (`
	`-  SELECT p.date`
	`-  FROM purchase as p, product as pr`
	`- WHERE p.pID=pr.pID AND pr.pName = 'Onions' )`
	`- GROUP BY c.cID, c.cName`
		semantic error	48

15	`SELECT *`
	`FROM customer as c, purchase as p, product as pr`
	`- WHERE p.cID = c.cID AND p.pID=pr.pID`
	`+ WHERE p.cID = c.cID AND p.pID=pr.pID AND cName='Julian'`
		semantic error	2

Attempt	Query	Correctness	Edit distance

16	`-  SELECT *`
	`+  SELECT c.cID, c.cName`
	`  FROM customer as c, purchase as p, product as pr`
	`-  WHERE p.cID = c.cID AND p.pID=pr.pID AND cName='Julian'`
	`+  WHERE p.cID = c.cID AND p.pID=pr.pID AND pr.pName = 'Coffee'`
	`+  AND (c.cID) IN (`
	`+  SELECT c.cID`
	`+  FROM purchase as p, product as pr`
	`+  WHERE p.pID=pr.pID`
	`+  AND pr.pName = 'Onions'`
	`+  AND (c.cID,p.date) IN (`
	`+  SELECT c.cID,p.date`
	`+  FROM purchase as p, product as pr`
	`+  WHERE p.pID=pr.pID AND pr.pName = 'Coffee' )`
	`+  )`
	`+  GROUP BY c.cID, c.cName`
		semantic error	47

Table 6: Attempts 12 through 16 for participant 15 on exercise 2. The difference between subsequent queries is annotated and highlighted with red and green colors. The edit distances are noted in the right-most column, and reflected in the colored lines. Attempts 14 and 16 show particularly high edit distances. This example shows that participant 15 temporarily changed tactics from approach A in attempt 12 and 13, to approach B in attempt 14 and 15, and finally back to approach A in attempt 16.