Identifying SQL misconceptions of novices: findings from a think-aloud study

Miedema, D.; Aivaloglou, E.; Fletcher, G.; Ko, A.J.; Vahrenhold, J.; McCauley, R.

Identifying SQL Misconceptions of Novices: Findings from a Think-Aloud Study

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

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

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

ABSTRACT

SQL is the most commonly taught database query language. While previous research has investigated the errors made by novices during SQL query formulation, the underlying causes for these errors have remained unexplored. Understanding the basic misconceptions held by novices which lead to these errors would help improve how we teach query languages to our students. In this paper we aim to identify the misconceptions that might be the causes of documented SQL errors that novices make. To this end, we conducted a qualitative think-aloud study to gather information on the thinking process of university students while solving query formulation problems. With the queries in hand, we analyzed the underlying causes for the errors made by our participants. In this paper we present the identified SQL misconceptions organized into four top-level categories: misconceptions based in previous course knowledge, generalization-based misconceptions, language-based misconceptions, and misconceptions due to an incomplete or incorrect mental model. A deep exploration of misconceptions can uncover gaps in instruction. By drawing attention to these, we aim to improve SQL education.

CCS CONCEPTS

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

KEYWORDS

SQL, novice, error, misconception

ACM Reference Format:
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 2021), August 16–19, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3446871.3469759

ICER 2021, August 16–19, 2021, Virtual Event, USA
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8326-4/21/08.
https://doi.org/10.1145/3446871.3469759

1 INTRODUCTION

Databases and the Structured Query Language (SQL) are core topics in Software Engineering and Computer Science curricula in higher education [33]. While the syntax of SQL is relatively simple in comparison to most programming languages, SQL is not a trivial language to learn. Several works have reported on common errors in the SQL queries written by students and novices, identifying various categories such as syntax errors, logical errors, semantic errors and complications [3, 8, 21, 31, 32, 34]. In the area of research surrounding SQL education, the main focus has been on the errors made, without expanding on the underlying causes or the misconceptions that novices may hold. The identification of misconceptions is an important first step towards devising instructional approaches that address students' difficulties. In the area of programming education, on the other hand, misconceptions have been well studied. Even though SQL is a query language, not a programming language, parallels between the two can be drawn.

Early works on programming misconceptions identified them as conceptual bugs in how novices program and understand programs [20], difficulties of learning to program, and errors based on the misapplication of analogies [7]. Since then, a large body of work has focused on misconceptions on introductory programming, and especially on imperative and object-oriented programming languages, identifying a wide range of errors in their conceptual understanding [9, 24]. One of the causes of difficulties and misconceptions that has been widely reported on is the misapplication of students' prior knowledge [12, 24]. When considering SQL education, relevant prior knowledge may come from several domains. Mathematics, set theory and relational algebra are some possible prerequisites for learning SQL and may thus influence learning of the material. Moreover, SQL borrows keywords from natural language, and could thus be susceptible to linguistic transfer [7, 9]. Prior exposure to other programming languages has also been found to cause misconceptions [25], for example due to the differences in the notations and their use, and SQL learners will commonly have been taught programming as part of the curricula they they follow. These different types of prior knowledge could be potential causes of misconceptions leading to SQL errors that have been documented in prior work. Early research on the causes of SQL errors hypothesized about five underlying causes: working memory overload, absence of retrieval cues, procedural fixedness, incorrect procedural knowledge, and misperception [27]. Recently, errors in the SQL queries submitted by students were mapped to these five causes by Taipalus [31], without however utilizing qualitative input from the students.

In this paper our goal is to understand the difficulties that novices face when formulating SQL queries and the underlying causes of their errors. The research question that we are aiming to answer is: *What are the misconceptions of SQL novices that cause difficulties or errors during query formulation?* To this end, we ran a think-aloud study in which we asked 21 students to solve query formulation problems. Similarly to other works identifying student misconceptions (for example, [13, 28]) a think-aloud study allows us to collect qualitative input on the students' thought process. Using the transcripts along with the students' notes, we identified the underlying causes of their errors which we categorize into four top-level categories: misconceptions based in previous course knowledge, generalization-based misconceptions, language-based misconceptions, and misconceptions due to an incomplete or incorrect mental model. To the best of our knowledge, this is the first study on the causes of SQL errors based on qualitative input on the students' thought process.

2 RELATED WORK

SQL education research has mainly focused on the errors made by novices, without expanding on the underlying reasons. Some previous work did explore causes by looking at human cognition and task analysis. One of those is Smelcer, who developed a model to explain known errors [27]. Unfortunately, their list of errors is very short, and focuses on basic errors such as omissions and misspellings. They list five underlying causes: working memory overload, absence of retrieval cues, procedural fixedness, incorrect procedural knowledge, and misperception. Misperceptions according to Smelcer are task-related, such as perceiving 'lc' as 'k', and thus lead to misspellings. This is largely unrelated to the misconceptions we discuss in this paper.

More recent work has expanded on the basic types of errors beyond omissions and misspellings. A typical split is for studies to focus on syntax errors [1], semantic errors [3, 8], or both [2, 21]. In their discussion of semantic errors, Brass and Goldberg distinguish between two types of answers [8]. Those queries for which we need to know the question to see whether they are correct, and those where the query is incorrect regardless of the question (for example due to a conflicting WHERE clause resulting in an empty answer). Taipalus and Siponen also use these two classes: they call the former category *logical errors*, and the latter *semantic errors* [34]. In addition, they add *complications* to their categorization. Complications are those unnecessary additions that muddle understanding of a query, but do produce the correct result table.

To get a better insight into which errors are more significant than others, Taipalus and Perälä consider persistent errors in student work. They define persistent errors as those that are present in the student's final answer [32]. They found that logical errors and complications had the highest incidence, and that syntax and semantic errors occurred least in final attempts. As a recap, logical errors are those that we can only evaluate as incorrect once we know the question the query tries to answer. This means that students wrote a query that in principle could be correct, but does not match the question they were asked. It would be interesting to take the students' perspective into account, to see where the mismatch between the question and query formulation occurs. This is where misconceptions come into play.

Research into misconceptions is one of the essential areas to understand when aiming to improve education. If we want to help our students learn to use SQL more effectively, we need to know the nature of the misunderstandings that result in difficulties or erroneous code being written. Unfortunately, there is not much work on misconceptions in SQL so far. In one paper we know of, Taipalus [31] discusses causes that could be behind the persistent errors explored in [32]. They map the errors to four cognitive explanations as introduced by Smelcer [27]. Unfortunately, this mapping is speculative [31], as they only use the plain text query as submitted, without qualitative discussion with students.

On the other hand, there is ample qualitative work on misconceptions in programming. Although SQL is a query language, not a programming language, parallels can be drawn. In our work on querying misconceptions, we draw from the research on programming misconceptions.

Misconceptions on introductory programming have been categorized into difficulties in syntactic knowledge, conceptual knowledge, and strategic knowledge, referring to knowledge about planning, writing, and debugging programs [24]. Towards identifying misconceptions, novice programmer errors have previously been categorized in syntax errors, semantic errors, and logic errors [17]. It was recognized early that an important source of misconceptions are properties that students assume of the machine that executes their code [7] or the superbug that Pea recognized as the assumption that "there is a hidden mind somewhere in the programming language that has intelligent, interpretive powers" [20]. Research on programming misconceptions has focused on the understanding of concepts in imperative and object oriented languages [15, 28] – an extensive list of misconceptions in presented in [29]. Apart from languages commonly taught in CS1, recent work on misconceptions has focused on visual programming languages used by younger learners [30], functional programming languages [11], as well as difficulties with the understanding of data structures [38].

Misconceptions have been considered as "faulty extensions of productive prior knowledge" that should be identified and refined rather than replaced [12]. In programming, students' prior knowledge, especially from natural language and mathematics, has been found to be transferred and cause misconceptions [9, 24]. Linguistic transfer is especially relevant to our work on SQL errors, since programming keywords borrowed from the English language can have ambiguous or different meaning [7, 22], such as the word *and*, which is a conjunction in natural language but a Boolean operator in programming. It has been found, for example, that novice programmers believed that an *if* statement was continuously actively waiting for the Boolean condition to be true, as used in natural language [20]. Several works have attributed misconceptions on the misapplication of prior knowledge on mathematics, especially algebraic notation. It has been observed, for example, to cause confusion on variable assignment statements [4] and affecting the understanding of execution sequences [28]. Prior exposure to different programming languages has also been found to cause misconceptions [25], for example due to the differences in the notations and their use for the definitions of variables and arithmetic operators.

Question	Summary of answers
What languages can you program in?	Java, C#, Python, JavaScript, C++, R
Have you been introduced to functional programming?	13/21
Have you studied the Relational Algebra?	17/21
Have you studied Set Theory?	14/21
Do you have work experience in the industry with programming- or query languages?	5/21 (all programming experience)
How would you rate your SQL proficiency? (1- Extremely novice, 10 - Complete expert)	mean: 5.35, standard deviation: 1.7
What interface do you use for writing SQL queries?	DataGrip, Jupyter, terminal, Microsoft Access, web interface, text editor with syntax highlighting.

Table 1: Questions on prior knowledge and experience of the participants.

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

Table 2: The database schema used for the questions in Table 3. Underlined attributes together form the primary key for each table.

With this knowledge on SQL errors and programming exercises, we aim to further the work in SQL education by identifying learners' difficulties and misconceptions. In contrast with the work by Taipalus [31], we ran a qualitative think-aloud study in which we asked students to solve query formulation problems. Based on the students' processes, we hypothesize on the underlying causes for their errors. In the following sections we elaborate on our study and findings, and present a first SQL misconceptions categorization.

3 METHOD

To collect information on the errors that SQL novices make and their underlying reasons, we ran a user study of semi-structured, think-aloud interviews. In the interviews, we presented participants with query formulation and evaluation problems of increasing difficulty.

The study took place in the Netherlands, through two universities and one high school. The interviews were held via Skype due to shelter-in-place measures. Participants wrote their notes and answers on paper, which they showed in front of the webcam. After the interview, they submitted pictures of their notes and answers to the researcher. The interviews were scheduled to take approximately 30 minutes. No compensation was provided.

3.1 Materials

Questionnaire Before the start of the interview, we asked the participants to asynchronously fill in a questionnaire to give us some insight into their knowledge of SQL and related subjects. The questions can be found in Table 1.

Interview questions The question pool for the think-aloud interview is available in Table 3, with the schema for the database represented in Table 2. We will refer back to these questions when discussing our results. The questions were designed to capture all basic SQL concepts that novices should be familiar with, while staying within the designated time frame for the study. In Table 3 we also mention *covered concepts*. These are based on the seven types of SQL queries defined by Ahadi et al. [2].

The starting question was chosen based on the self-reported SQL score. For participants with score of 5 or lower, we started from question 1. All others started with question 3. We found that many participants scored themselves either too low or too high, which meant that we sometimes deviated from this procedure by going back or skipping questions, to get the most out of the interview.

3.2 Participants

For our participant pool we were looking for students who had recently learnt SQL. For recruitment, we reached out to database course instructors the authors knew. Interested instructors forwarded our interview invitation to their students, who could then register for a time slot.

We had 21 participants, of whom 3 were female and 18 male. No participants reported disabilities. Only two of the participants in our study were native English speakers; the others had different first languages. To make the study available to all participants, the questions were posed in English. As the native language of the interviewer and most of the participants was the same, 15 of the interviews were held in that language.

Regarding their education level, our participants fell in two categories. One group (n=4) were high school students who were taking Informatics as part of their curriculum, had been taught and had just been tested on SQL by their teacher. The second group were university students from two different universities (U1: n=4, U2: n=13) who had taken one course on Databases before the interview, which included 2 lectures on SQL. Most of our participants could be called novices with regard to SQL.

Table 1 summarizes the reported prior knowledge and experience of the participants. For SQL proficiency, they graded themselves on average as 5.35 (on a scale of 1 to 10), with a standard deviation of 1.7. All participants knew how to program in at least one programming language, with most people knowing two or more programming languages.

3.3 Procedure

All interviews were held online via Skype by the first author, who was not a member of the teaching team of the databases courses. The participants were sent the questionnaire and informed consent form before the start of the interview, and could return pictures or scans of these documents. The interviews started with the participants being introduced to the ideas behind the study and to the think-aloud protocol. We also checked whether they had pen and paper ready, and we ran quickly through the main points of the informed consent to allow for questions. Then the interviewer let the participant know that a recording would be started, and the study began.

	Question type	Question	Covered concepts	Tried by
1	Execution	What is the result of Query 1? What is the natural language equivalent?		11
2	Formulation	List all IDs&names of customers living in Eindhoven.	Simple query	12
3	Formulation	List all pairs of customer IDs who live on a street with the same name but in a different city.	Self-join	21
4	Formulation	List all customer IDs, dates and quantities of transactions containing products named Apples.	Natural join	21
5	Advanced formulation	Find the names of all inventory items that have a higher unit price than Bananas.	Single subquery	16
6	Advanced formulation	Return a list of the number of stores per city	Group By	7
7	Advanced formulation	Return the stores table ordered alphabetically on city.	Order By	5
8	Execution	What is the result of Query 2? What is the natural language equivalent?		6
9	Advanced formulation	A store-chain consists of at least two stores with the same name but different IDs. Find the names of the store-chains that on average sell product in quantities of more than 4.	High complexity	3

Table 3: The pool of questions available for the interview.

```
SELECT c.cID, s.sID, c.city
FROM customer AS c,
     store AS s
WHERE c.street = s.street
AND c.city = s.city
```

Query 1: First execution trace question.

```
SELECT AVG(price), pName
FROM product AS p,
     transaction AS t
WHERE p.pID = t.pID
GROUP BY pName
HAVING AVG(price) > 1
```

Query 2: Second execution trace question.

We introduced the database schema and some example data (see Figure 1) that were the basis of the query formulation problems, and gave the participants time to study it and ask any questions they may have. Depending on their self-reported SQL skill-level, we started either with question 1 or question 3 of Table 3. Students were asked to read each question out loud and express their thoughts, to encourage discussion on their problem solving process.

During query formulation, and especially upon the occurrence of mistakes, the participants were asked to elaborate on the how and why of their approach, to try to uncover the reasons behind the errors. Sometimes, the query would be corrected by the students (who were asked to leave the erroneous answer visible), sometimes they made new mistakes and the discussion would start again.

In general, the study was ended after the designated time had passed and no more problems were presented, except in cases where participants suggested we continued longer. Then, the participants were asked if they had any questions or remarks. The recording was stopped, and the interviewer asked the participant to submit pictures of their notes and queries. Finally, the interviewer asked the participant whether they wanted to be kept up to date with the research, and the participant was thanked for their participation.

3.4 Data processing

For data analysis, we used the notes of the participants on their intermediate and final answers to the interview questions in Table 3 and the transcribed interviews.

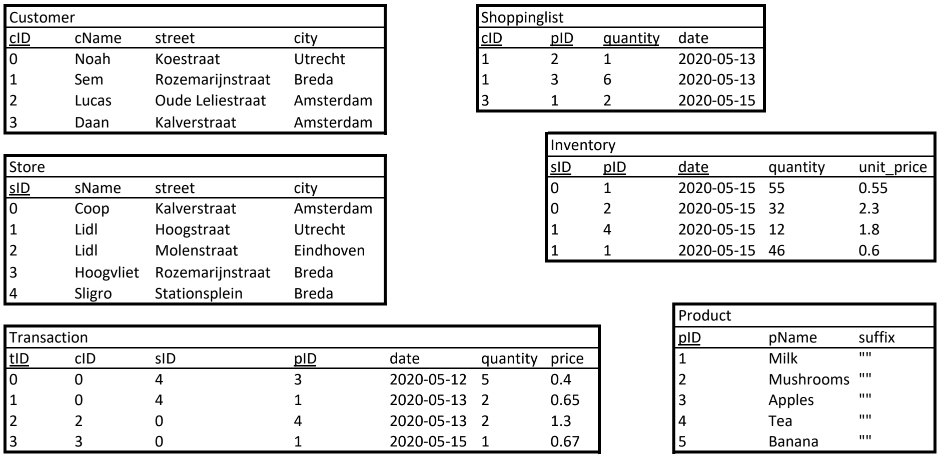
The first step in our analysis was to extract all errors, both in intermediate and final attempts. We made a list of all distinct errors that occurred, and classified each error according to the SQL keyword they were associated with. Those errors were then organized according to categories of SQL errors identified in existing work [8, 34]. The error categorization is presented in Section 4.

Following the identification of the errors, we used the transcribed interviews to explore the causes of the errors. For every identified error in the final or intermediate query result, we searched for indications of its cause in the thought process as it was expressed by the participant. The causes were identified either in the thought process leading to the errors, or in their answers to clarification questions of the interviewer. We followed a data-driven approach, generating appropriate labels according to the expressed misconceptions [6]. Once this process was finished, the three authors separately looked at the labels, along with representative quotes for each, and independently came up with misconception categorizations. During the categorization we took into account the interference and transfer of existing knowledge in the development of new concepts. The researchers then compared and discussed their categorizations, and agreed on four top-level categories: misconceptions based on previous course knowledge, generalization-based misconceptions, language-based misconceptions, and misconceptions due to an incomplete or incorrect mental model.

4 IDENTIFIED SQL ERRORS

After the interviews, the first intermediate step towards exploring students' misconceptions was the labeling the errors they made in their queries. For this we used all intermediate and final queries as written by the participants.

As our focus is on exploring the misconceptions that lead to these errors, we will not present this full list of errors in this paper. Instead, we grouped the errors into categories and report on this summary. We split up semantic and logical error categories into smaller groups, as we believe these categories are too coarse, and thus not descriptive enough. We used the following categories: syntax issues, incorrect or missing table/column, incorrect or missing keyword, returning incorrect results, issues with the schema, alias problems, contractions, and complications.



Customer

cID	cName	street	city
0	Noah	Koestraat	Utrecht
1	Sem	Rozemarijnstraat	Breda
2	Lucas	Oude Leliestraat	Amsterdam
3	Daan	Kalverstraat	Amsterdam

Shoppinglist

cID	pID	quantity	date
1	2	1	2020-05-13
1	3	6	2020-05-13
3	1	2	2020-05-15

Inventory

sID	pID	date	quantity	unit_price
0	1	2020-05-15	55	0.55
0	2	2020-05-15	32	2.3
1	4	2020-05-15	12	1.8
1	1	2020-05-15	46	0.6

Store

sID	sName	street	city
0	Coop	Kalverstraat	Amsterdam
1	Lidl	Hoogstraat	Utrecht
2	Lidl	Molenstraat	Eindhoven
3	Hoogvliet	Rozemarijnstraat	Breda
4	Sligro	Stationsplein	Breda

Transaction

tID	cID	sID	pID	date	quantity	price
0	0	4	3	2020-05-12	5	0.4
1	0	4	1	2020-05-13	2	0.65
2	2	0	4	2020-05-13	2	1.3
3	3	0	1	2020-05-15	1	0.67

Product

pID	pName	suffix
1	Milk	""
2	Mushrooms	""
3	Apples	""
4	Tea	""
5	Banana	""

Figure 1: The sample data presented to the participants. The database schema is based on a group of supermarkets.

The presentation of the errors below is a first step in the analysis of our data. The labeling of these mistakes helps us to order the data before we examine their underlying causes in Section 5.

4.1 Syntax issues

Syntax errors is a common error category, present in various other papers on SQL errors [1, 21, 34–36].

As is to be expected when writing in a language as strict as SQL, many syntax errors occurred during the interviews. Some of these were very basic, such as forgetting commas and brackets, as well as the quotes around string data. These can be attributed to distractions and sloppiness. Others were more involved, and show some problems with understanding how SQL works. A basic example is incorrect writing of operators. For 'not equals', both <> and != are valid ways, but some participants wrote ≠ or 'IS NOT'. For 'equals', SQL requires =, but some participants wrote ==. Other errors concerned the usage of comma and AND, which are not interchangeable. Participants substituted comma for AND, or skipped the AND or comma altogether.

Another subcategory of syntax issues is that of incorrectly ordered keywords and clauses. We found participants incorrectly placing the GROUP BY keyword at the beginning of the query, and reordering SELECT, FROM and WHERE.

4.2 Incorrect or missing table/column

A similar problem to the missing or incorrect keywords also happened for the data itself. Some participants used incorrect tables or confused table- and column names. Others projected only on part of the required columns, or forgot to include a table for self-join.

Ahadi et al. collect errors in this and the previous category under 'omission errors' [3]. From Presler-Marshall et al. this category may include *wrong subclauses in the WHERE clause*, *missing JOIN* and *column mismatch*. It is also related to Taipalus et al.'s undefined database object and projection errors [34], and Welty et al.'s minor operand error [36]. Errors regarding attributes and their types are categorized by Welty as complex errors [35]. This category does not occur in Brass et al.'s list of semantic errors [8].

4.3 Incorrect or missing keyword

Other researchers classify these errors as syntax errors [8, 34], but we feel that there is a difference between missing commas and other minor elements on the one hand, and complete missing clauses on the other hand. Presler-Marshall et al. include this as missing or extra operator in their semantics category [21]. We include the switching up of keywords in a separate category. Sometimes there were mix-ups, but there were also some creative inventions by our participants.

Some participants also made mistakes based on synonyms: SUM versus COUNT and SORT versus ORDER BY. We also had some cases of missing keywords: dropping BY from ORDER BY, missing ON for explicit JOINs, and missing FROM. The questions we posed were all retrieval queries. Nevertheless, one participant wanted to use INSERT instead of SELECT. Finally, one participant thought that for taking the first item of a list, they should use DISTINCT.

4.4 Returning incorrect results

This category includes queries that were syntactically correct, but would result in incorrect results. Some queries include incorrect operators (= instead of !=) or incorrect functions, such as adding COUNT when not required. Incorrect results also happened because of issues with the JOINs, such as omitting the JOIN condition, incorrect JOIN conditions, and comparing unmatchable items (street = city). Projections also went wrong regularly. Participants projected in the wrong order, on extra columns, or missed columns.

Taipalus et al. call these logical errors in the subcategories on operators and joins [34]. Similarly, it includes some errors listed by Welty and Stemple: minor language error, minor substance error and major substance error [36]. Incorrect results also includes some errors reported by Presler-Marshall et al.: wrong values in where, wrong ordering, wrong operator in WHERE [21]. It does not occur in Brass et al.'s list of semantic errors [8].

4.5 Issues with the database schema

This category is related to the process of the participants, and thus has not been reported in previous research before.

As we discussed in the related work section of this paper, problems with query formulation can arise from the need to remember the database schema. We found various mistakes that are closely related to the schema, most of which involve confusion on which table or attribute to use. Other schema issues include confusing a table name for a column name, and misspellings in both the table and column names.

4.6 Alias problems

One thing that many participants struggled with were aliases. It seems that students do not always have a correct grasp of what aliases are meant for, and when to use them. This relates both to alias syntax (writing the alias behind the column name, using an alias but not defining it), and to its applicability. For example, some queries contained ambiguous column names as a result of missing aliases. Other queries included aliases when they were not required.

Taipalus et al. classify some of these mistakes under syntax errors and others in complications [34]. From Presler-Marshall et al., alias problems are captured under column reference error [21]. This category is not included in Brass et al.'s list of semantic errors [8].

4.7 Contractions

This is a set of errors that overlaps with syntax errors, but reflects cases where our participants wanted to write the queries in shorter form, even if this was not syntactically allowed (e.g., Query 3).

Some participants had lost the basic SQL syntax, and made mistakes in which they combined or confused various parts of the queries. One participant merged the SELECT and FROM clauses, and wrote: SELECT customer AS a, customer AS b, where *customer* is a table name in the schema. Something similar happened to the SELECT and WHERE clauses, where participants projected on items that should be selected on.

Other confusions included subqueries, where conditions were not placed on the correct level of nesting, and aggregation. Aggregation was already shown by other researchers (see [8]) to be difficult. The main aggregation issue we found is on where GROUP BY is required, which resulted in several incorrect answers.

```
SELECT customer AS a, customer AS b
WHERE a.street = b. street
AND a.city <> b. city ;
```

Query 3: Participant 16, question 2, an example of a contracted query.

4.8 Complications

Complication is a common category supported by previous literature [8, 34] that describes those parts of the query that do not alter the answer, but complicate the query formulation and make the query more difficult to interpret and check. Examples include GROUP BY on singleton groups, unnecessary JOIN, or using aliases when not necessary.

Some of these complications were purely visual. For example, one participant wrote out all column names instead of using *. Other participants wrote unnecessary brackets, renamed columns, or defined aliases but did not use them. Another complication for aliases was impractical naming. Most participants had learned to use the first letter of the table name. Other participants had not learned this, and instead used A, B, C as aliases. This significantly reduces readability, and for these participants we saw some swapped aliases, where incorrect sets of alias and column were combined.

Other complications were functional, and would result in extra processing of the query. This includes applying DISTINCT on a result that contains no duplicates, and adding an extra condition that is already captured by something else in the query. Another process we saw a few times was participants including tables in the FROM clause if they needed to return this table's primary key, regardless of whether this was also included as foreign key in another table. This resulted in extra JOINs that were not required.

5 RESULTS

In this section we present the misconceptions that were revealed from the analysis of the SQL query formulation process of our participants, organized into four high-level categories: misconceptions based on previous course knowledge, generalization-based misconceptions, language-based misconceptions, and misconceptions due to an incomplete or incorrect mental model.

5.1 Misconceptions based on previous course knowledge

SQL is a topic in most university programs in Computer Science. However, even in those programs, it will not be the first thing taught to the students. In the authors' institutions, the introductory Databases course is taught only in the second year. Additionally, students place knowledge in the context of what they already know. Anything they have learnt so far can influence new knowledge. The cause of several of the errors that our study participants made was the misapplication of previous course knowledge from mathematics, programming, and databases.

5.1.1 Mathematics.

Believing ≠ is a valid comparison operator in SQL. Students take knowledge from mathematics and use this as information on what operators are valid. For example, some students wrote queries containing ≠, instead of != or <>. The latter two are a valid operation in SQL, whereas the first is not. However, ≠ is often used in Mathematics courses, and the students are familiar with it. Several participants mentioned that they did not remember which symbol(s) to use for non-equality, and apparently ≠ is most accessible in their memory.

5.1.2 Programming.

SQL is a query language, and therefore close to programming languages and susceptible to knowledge transfer. Our participants had experience with various programming languages, such as Python, Java, C# and JavaScript. Therefore, some misconceptions we found are similar to those in programming misconceptions literature.

Confusing == and =. Students use these interchangeably in various contexts, as has been shown by research on programming misconceptions. We also found this confusion in our SQL experiments. This mistake makes sense, as in programming languages = is for assignment and == for comparison. However, in SQL, == is not used. Instead, = is used for comparison, and there is no assignment.

```
WITH u AS (SELECT i.sID, i.date, i.unit_price
    FROM inventory AS i, product AS p
    WHERE i.pID = p.pID AND
    p.pName = "Bananas")
SELECT p.pName
FROM inventory AS i, product AS p
WHERE u.sID = i.sID AND
    u.date = i.date AND
    i.unit_price > u.unit_price AND
    i.pID = p.pID
```

Query 4: Participant 3, query 5, misunderstanding the scope of the query.

Misunderstanding the scope of elements in a query. As discussed in the related work, some misconceptions within programming have to do with the scope of the code. Issues that have been shown to cause confusion include variable assignments and when if-statements are checked. In a similar vein, we found two errors that have to do with a misunderstanding of the scope in SQL.

First, students defined a view by writing a query with a WITH clause. However, a view still needs to be called in the main query, in order to be used appropriately. In the case of Query 4, the student thought that as they had defined the query, this would be sufficient.

Second, a common error occurred in the self-join. A self-join query is often used to retrieve pairs of entries from the same table, in which case the same table needs to be called twice with different aliases. However, various participants only called the table they wanted to use once (see an example in Query 14), then tried to figure out how to project a distinct pair. This is impossible without calling twice and thus led to unfinished attempts. Participants expressed their confusion on how to solve this problem, either because they did not understand that they had to use the same table twice, or because they did not know the notation.

5.1.3 *Databases.*

Typically, students following a course on Databases are taught Relational Algebra first, with SQL taught afterwards. As the two are closely related, it is likely that students apply knowledge of Relational Algebra to SQL, which may lead to mistakes. In our population, this applied to 13 of our 21 participants.

```
SELECT B.pName
FROM inventory A, product B
WHERE A.pID = B.pID
AND B.unit_price >
    (SELECT C.unit_price
    FROM inventory C, product D
    WHERE C.pName = "Banana"
    AND D.pID = C.pID)
```

Query 5: Participant 10, question 5.

Believing DISTINCT takes the first item of a list. For question 5 in Table 3, there was a follow-up question to gather more insights in the understanding of subqueries. Suppose a user writes a subquery to find the price of 'Bananas'. Now, as pID by itself is not a primary key in the inventory table (see Table 2), the subquery may return more than one correct price. The follow up question to the students was: "How do you select one price, to make sure the query does what you want?" Most students suggested using MIN, MAX or TOP to select one price. Participant 10 wanted to use DISTINCT, and believes that this will make sure the subquery only returns one result. For reference, this student's query is available in Query 5. Here is what they have to say about solving this problem:

> "If I would have used a DISTINCT before, on the unit price, then you'd have taken the first unit price and compared it with the unit price of the primary query [...] If I used DISTINCT over this C.unit_price, I'd have given only one unit price and probably that would have been the first one, because that is what it will find first." - Participant 10

When the interviewer asked for clarification on whether this would still be true when the Bananas have different prices, the student said:

> "Yes, only in the case of [...] if they have two prices. If they have the same [price] then it does not make a difference also right?" - Participant 10

This quote clearly illustrates the student's misconception. We believe that such a misconception can arise from students being presented with very selective examples only.

Believing ≠ is a valid comparison operator in SQL. This misconception, also mentioned in Section 5.1.1, can be caused by various sets of existing knowledge. We already discussed the possibility of a relationship with mathematics, but another option is that students apply knowledge from the Relational Algebra, where ≠ is regularly used.

5.2 Generalization-based misconceptions.

One large source of mistakes that we identified is the generalization of previously learned, SQL-related items. This might involve examples in a book or other course material, or a query that worked out previously but might not work in all cases. Within this top-level category, we consider two types: templates and consistency.

5.2.1 *Applying an incorrect query template.*

Templates are (parts of) queries that students have used before, with success, leading the students to reuse them.

The first template we found is to equate primary keys. Typically, when using a Cartesian product to create a JOIN, one would take an attribute from both of the tables and equate them. During our study, we found evidence of students applying such a template. However, this is not correct in all cases. When looking for all permutations or trying to find distinct pairs, as we ask for in question 3, no keys should be equated.

Another indication of the use of a template is to look at what items should be projected, which of these are primary keys, and then include the tables for each of these primary keys in the query. This might not be necessary, as tables may be connected by foreign keys. One could argue that this is not necessarily incorrect, just inefficient, but it leads to longer queries and thus decreases readability.

```
SELECT city, COUNT(city)
FROM store
GROUP BY city;
```

Query 6: Participant 15, question 6, counting the column they group on.

Finally, a concrete template that we found being misapplied was regarding the COUNT keyword. For question 6 in Table 3, *Return the number of stores per city*, one participant counted the same column they grouped on, as can be seen in Query 6. The question implied the counting of the store IDs (the primary key in the table). The interviewer asked whether grouping by an attribute and then counting it would result in a count of one. The student expressed that he was sure that this was not the case, because they had used this type of query a lot recently, for a project. However, they could not explain why this worked. When the researcher hinted that they should count a different column, the student said:

> "Then I don't really know [which one]." - Participant 15

This indicates that they applied a template without being able to reason why.

5.2.2 Misunderstanding SQL syntax and its internal (in)consistency. The consistency category is similar to the templates one, but instead considers SQL syntax elements that students has difficulty with because they are applied differently in different cases.

First, there is the usage of commas. In both the SELECT and FROM clauses, commas are used for the separation of items. However, the WHERE clause requires the usage of AND to separate conditions. This inconsistency in the SQL syntax was difficult to distinguish for some of the participants and led to them using a comma instead of an AND in the WHERE clause.

Second, a consistency misconception was identified for aggregates. In SQL, aggregates are only allowed in the SELECT or HAVING clause. However, some participants used them in the WHERE clause. An aggregate cannot be applied here, as there is no group over which to aggregate, but that is not clear to students. We believe that part of the error stems from WHERE and HAVING being locations to place conditions. Everything that can be used in WHERE can be used in HAVING, but not vice versa, which is an inconsistency that caused confusions and led to mistakes.

```
SELECT cID.transaction, date . transaction ,
    quantity . transaction
FROM transaction AS t, product AS p
JOIN product
ON transaction .pID = product .pID
WHERE product.pName = "Apples";
```

Query 7: Participant 21, question 4, tablename behind the attribute in SELECT.

Another consistency misconception we found relates to the use of aliases. In this specific case, we discuss the inconsistency between defining the alias behind the relation, and calling the alias in front of an attribute. In the SELECT clause, users should write alias.attribute, whereas for definition it is relation (AS) alias. The option to use the keyword might make this latter case more salient, which leads to more errors in the former case. Three of the interviewed students were affected by this inconsistency, making mistakes such as in Query 7 and Query 8.

```
SELECT pName p
FROM product p
WHERE pID IN (SELECT pID
    FROM inventory
    WHERE unit_price > 2.3);
```

Query 8: Participant 4, question 5, alias behind the attribute in SELECT.

Finally, there is the question of semicolon usage. Students typically learn that every query should be terminated with a semicolon. Participant 19 took this to heart, as we can see in Query 9. However, semicolons should not be used in subqueries. Depending on the number of examples shown to the students, and the method of teaching, this exception may not be clear to all students. Thus, they may consistently apply the rule they have learned: every query should be terminated with a semicolon.

```
SELECT p.pName
FROM inventory AS i
WHERE i.unit_price > (SELECT i. unit_price
    FROM inventory AS i
    JOIN product p ON i.pID = p.pID
    WHERE p.pName = "Banana";);
```

Query 9: Participant 19, question 5, using a semicolon in the subquery.

5.3 Language-based misconceptions.

Although clauses in SQL can be read as something close to Natural Language, inherently, there is still a big difference between the two. Therefore, language can be a source of misconceptions too.

Believing IS NOT is a valid comparison operator in SQL. In Section 5.1.1 we already mentioned that many students did not remember how to write down inequality. Besides the option of writing ≠, some students in our study wrote IS NOT. This would make sense if we look at the keywords available in SQL, as IS exists too. IS may not be usable for asserting (in)equality in the WHERE clause, but in natural language it is, which led to students using IS or IS NOT instead of = or !=/<>.

Remembering only the core part of a keyword. A second language aspect of SQL is that many keywords include prepositions: INSERT INTO, GROUP BY, JOIN ON. In various questions, we found that students dropped these prepositions during query formulation, and thus only wrote INSERT, GROUP and JOIN (see Query 10). From a Natural Language standpoint, these keywords might be the main part of the clause, as the prepositions are not required to make sense of the query when reading it. Unfortunately, syntax-wise the clauses don't work without the addition of the preposition. The clauses may have been mistakenly stored in memory just by their 'main' part, disregarding the prepositions.

```
SELECT transaction.cID,
    transaction .date ,
    transaction .quantity
FROM transaction
JOIN transaction .pID = product.pName
WHERE pnamed = "Apples";
```

Query 10: Participant 18, question 4, various issues.

Using synonyms of SQL keywords. The error category of synonyms used while formulating queries has been noted by various authors [19, 27]. We found the occurrence of synonyms in table- and attribute names, as well as in SQL keywords. The former indicates a lack of knowledge about the database schema used in the exercise, or a relation to the question asked such as in Query 10, but the latter may indicate a language-based misconception. One such example is shown in Query 11, where a student wrote SORT BY instead of ORDER BY. SORT is not a valid SQL keyword. We believe that the word sort is more commonly used as a verb than order, and thus may have been more accessible to the student during the study. This is interesting, as question 7 (see Table 3) actually did not mention sort, but instead used “ordered alphabetically”.

```
SELECT *
FROM store
SORT BY city ASC;
```

Query 11: Participant 14, question 7, participant used SORT instead of ORDER.

Usage of a non-native language. Finally, during the study we noticed some language issues. Many non-native English speaking participants translated the problems to their native language and reasoned in that language before translating back to English for SQL. For example, question 4 contained the word ‘quantities’, which indicated a column in one of the tables. Some participants translated the question to their native language, which changed the implications of the question. In the end, some of our translating participants (such as participant 20, Query 12) arrived at the conclusion that they needed to count something, and thus applied the COUNT keyword.

```
SELECT cID, COUNT(tID), transaction.date
FROM customer c
JOIN transaction  t ON c.cID = t.cID
JOIN product p on t.pID = p.pID
WHERE p.pName = "Apples";
```

Query 12: Participant 20, question 4, including the customer table when this is not required.

5.4 Misconceptions due to an incomplete or incorrect mental model.

Although all of the misconceptions we discuss above arise from incorrect mental models, some errors show this more explicitly than others. In this section we discuss all those SQL concepts that students struggled with because of incomplete or incorrect mental models.

```
SELECT cID, cName
FROM customer
WHERE (city = "Eindhoven");
```

Query 13: Participant 17, question 2, participant used extraneous brackets.

Using brackets as a crutch. First of all, we start with a general indication of incomplete mental models. Some of the students in this study included extra brackets where they do not make sense, such as in the WHERE clause (see Query 13). Although this is not explicitly incorrect as SQL can ignore these brackets, they do reduce readability. Also, the participant probably had a reason why they included these brackets. This might be because they associate WHERE with brackets, through subqueries. Existing work shows that participants use brackets to try to fix errors [18] instead of trying to find the actual mistake and fixing it. This shows a gap in the knowledge of the students.

Lacking knowledge on primary keys. Several participants indicated confusion with the use of primary keys in different situations. For example, when trying to connect two tables through a Cartesian product, one participant said:

> “I need to connect the two tables with a primary key. Or did I do that already? I’m not sure...” - Participant 8

In actuality, they had already written down the correct WHERE clause to appropriately connect the tables.

Furthermore, we saw evidence of this difficulty in the application of DISTINCT on attributes that were primary keys. When an attribute is a primary key, by definition it should only occur once in the data. Therefore, these keys are unique by definition and thus DISTINCT is not required. There were some participants who noted that it was probably not needed, but apparently were not sure enough to leave it off, which indicates a lack of knowledge on the effects of primary keys.

```
SELECT cID
FROM customers
WHERE street = street
AND city <> city ;
```

Query 14: Participant 18, question 3, an attempt at a self-join without a second table present.

Thinking of the DBMS as a black box. The self-join error, described in Section 5.1.2 under scoping, can also be explained by a different misconception, namely that of thinking of the DBMS as a black box. For example, after the participants had failed to include the second table for answering question 3 in Table 3, the interviewer hinted for some participants that they needed to include a JOIN, or another copy of the table. One participant, who had used ‘copies’ of the attributes in the SELECT and WHERE clause, said in response: “I’d ask you why..”, indicating that they could not see that they needed two copies. This means they thought the DBMS could resolve Query 14.

Another participant did not understand in which cases they needed to add JOIN conditions. Query 15 is their initial query. Participant 9 elaborates:

> “I’m not sure how they join. Or if they join. If they join on the pIDs automatically, or if I have to say something. [. . .] Because I’ve already filtered on the rows that have Apples, and I’m selecting from both.” - Participant 9

Taking an incorrect perspective. From the reasoning of the participants during the study, another misconception we found was that of an incorrect perspective. The majority of the participants approached each problem from the problem description. However, some participants focused on how the result table should look, without thinking about the implications for the query. One example of this is Query 10. This participant tries to join two tables but makes various mistakes and has selected two incompatible attributes, trying to compare a pID to a pName. When the interviewer asked why they did not use the same column twice (pID = pID), they discuss how pID is already covered as it is in the *transaction* table. Because the question requires them to find the pName belonging to each pID, they want to ‘move’ the query to *product*, where pName is encoded. The fact that transaction.pID is already included in the query, does not necessarily mean anything to the DBMS. In this case, it specifically inhibits answers, as the JOIN condition is incorrect. This participant thus approached the problem from the wrong perspective, leading to an empty results table.

Another example of the incorrect perspective happened for Participant 9. After reading the posed question, they took a look at the database schema to see which table(s) contained all the attributes they needed. As *shoppinglist* was one of the tables shown at the top of the schema, and it contained the required elements, they decided to compose the query using *shoppinglist* (see Query 15). However, the query stated that they needed to use the *transaction* table. As their perspective was incorrect, their query in turn was also incorrect.

```
SELECT s.cID, s.date, s.quantity
FROM shoppinglist s, product p
WHERE p.pName = "Apples";
```

Query 15: Participant 9, question 4, missing JOIN condition in Cartesian product.

5.5 Other observations

All participants made syntax errors. This includes omitting quotes around strings, incorrectly writing table and column names, omitting the semicolon, and using non-standard operators. Most of these syntax errors were likely due to lapses in focus and were usually subsequently fixed.

During our interviews we saw that participants were often confused. Major confusion arose around JOINs and aliases. Participants did not know when to use them, whether they had already completed this aspect of the query, what each type was called (Cartesian product, inner join, natural join), and so forth.

Another cause for confusion were query formulation problems where the data matching the question (that what would be included in the result table) is not included in the sample database. In those cases, students struggled significantly with writing the query because they would look at the data and not understand how to write the query if the data was not in there. For example, for problem 5 in Table 3, several participants concluded that no price for Bananas was available.

> “What is the unit price that I can use? In this case, no one has Bananas, so... There are no Bananas, so there is no point.” - Participant 8

This is interesting, as typically, we would only look at the schema (not the data) when writing a query. Finding the data (in this case the Banana price) is the goal of the query, so looking at the data itself seems unproductive. Moreover, checking whether data is present before writing a query is infeasible in normal situations. Databases are much too large to look through manually. Therefore, it is interesting to see that three of our participants engaged in this practice (participants 4, 8 and 15).

Another issue with the schema is that, instead of reading the question attentively, the participants would look at the attributes they need to return, and check in the database schema for which of the tables contain these elements. This disregards the meaning of the data completely. For example, in our schema, *shoppinglist* and *purchase* contain similar columns. However, if you blindly substitute one for the other, the meaning of the query completely changes.

On the other hand, some participants showed higher levels of insight by discussing primary keys and what this meant for the query formulation. For example, for question 5 in Table 3, some participants noted that date was part of the primary key for the inventory table. This means that these participants wrote a different query than those who did not notice this. Another aspect where primary keys play a role is on whether DISTINCT is required for solutions.

5.5.1 *Lack of knowledge or experience.*

Another noteworthy fact is the prevalent lack of knowledge or experience on various SQL concepts. These include GROUP BY, JOIN, aliases and subqueries. The issues with these concepts were so major for some participants, that it is unfeasible to uncover specific misconceptions.

For **GROUP BY**, we found issues with placement within the query, as well as confusion on when it is required. For example, when attempting to solve problem 7 from Table 3, which requires ordering a table, participant 7 also included a GROUP BY clause:

> “I think GROUP BY is very annoying. Sometimes you need it, and sometimes you don’t. In this case, I think ORDER BY is probably enough. So then I’d say to remove the GROUP BY, but I don’t know (I can’t test) whether it influences the answer. [...] As far as I know it can’t hurt.”
> - Participant 7

When we are looking for the reordering of a table, adding GROUP BY in most cases significantly changes the answer. In this case, grouping on a column and then ordering on the same column discards a lot of the rows of the table. It would be interesting to look at these situations in more detail in follow up studies.

```
SELECT c1.cID, c2.cID
FROM customer, customer
WHERE c1.street == c2. street
AND c1.city != c2. city
```

Query 16: Participant 12, question 3, using an alias without defining it.

For **aliases**, we found they were used inconsistently by our participants. We had participants defining aliases and not using them, or using aliases without defining them properly (Query 16). When discussing in more detail, we found that some participants thought that the DBMS looks at all column names, instead of only the ones of tables that are included in the FROM clause. One such example is participant 20. For problem 2 in Table 3, participants need to use only one table, and thus no aliasing is required. When we discuss this, they say the following:

> "The customer table has city, which is what you want to filter on. But because we also have city in the store table, you can't just put city in this query, because if you do that [the DBMS] does not know what to filter on. Therefore you need to specify that you want customer.city." - Participant 20

Other participants discuss that they are always applying them automatically, without thinking:

> "I'm always doing it automatically, but for one table I don't believe we have to do it like this." - Participant 13

On the other hand, there were participants who made it sound like aliases are optional, whereas in some cases they are required:

> "Sometimes it saves trouble, but sometimes it's just more complicated. It depends how complicated you're getting. It saves typing more, I suppose." - Participant 9

```
SELECT t.cID, t.date, t.quantity
FROM transaction t
WHERE t.pID IN (SELECT p.pID
    FROM product p
    WHERE pName = "Apples")
```

Query 17: Participant 4 using a subquery instead of a join.

The final concept that participants did not seem to have a good grasp on were **subqueries**. They were used even in cases where they did not make sense and made the problem solving and the understanding of the query more difficult.

Participant 4 decided to use a subquery for problem 4 in Table 3. This query formulation problem is relatively straightforward when using a Cartesian product (as they used for problem 3) or a natural join, but this participant did not use them. Instead, they wrote Query 8 and Query 17 using the IN keyword. This is a cross-over of lack of understanding of JOIN as well, but with a creative solution. The clarification that they did not know how to use subqueries came from Query 8. Here, they explained that they did not know how to have the subquery return a single integer. They should retrieve the price for a certain ID, but instead suggest to hardcode the price in their subquery.

> "I don't know how to do this. I did it before, it was one of the exercises in my assignment. Then it was also very difficult for me to extract the number." - Participant 4

Another misunderstanding on subqueries is shown by participant 5. About question 5 and subqueries returning results, they say:

> "The subquery returns the product where the name is Banana, so there is only one answer, but I will take the average because it requires aggregation. If I don't, [the DBMS] will see it as a list of answers." - Participant 5

This is not the case. If there is only one record in the subquery, leaving the aggregation out results in a valid query too.

6 DISCUSSION

It is already known from existing work in computing education that programming misconceptions are often caused by the transfer of students' prior knowledge from natural language, mathematics, and other programming languages [9, 24]. Our findings confirm that this is also the case for SQL, highlighting the areas where this transfer occurs and the errors that it causes. The identification of the faulty transfer of prior knowledge is the first step towards knowledge refinement and reorganization [12]. Several approaches have been proposed for addressing misconceptions in introductory programming [24], which could apply to SQL instruction. One of the approaches that has been lately demonstrated to be effective in facilitating transfer of revised knowledge are refutation texts, which explicitly state and refute misconceptions [5].

One of the most challenging concepts for our study participants, causing various problems in their query formulation process, were table joins. We believe that this might have its origin in the variety of methods available for combining tables. Participant 21, for example, combined two methods in one query (Query 8). Our overall impression is that joining is a confusing concept to many students. This has been shown by other researchers too. Kearns et al. write that "Students often struggle with the concept of the relational join, and they find it hard to visualize what joined tables look like." [14, p. 226]. More recently, Taipalus and Perälä state that "Join errors (LOG-2) were common in almost all multi-table queries, both all and final." [32, p. 202]. We also found students who had no problems with Cartesian products or natural join, but got stuck on the self-join. This has also been shown by Ahadi et al.: "62% of all students who could answer a natural join could not provide a correct self-join." [2, p. 204]. We unfortunately have not gained much insight into why joins appear to be such a problematic concept.

Our participants were also often led to errors because of taking an incorrect perspective in query formulation. Research in programming misconceptions has revealed that students often show difficulties in understanding the task and decomposing the problem [24]. In SQL this is especially hard because it is a declarative language. SQL query execution can thus not be traced in the same way as programs written in imperative programming languages can. In programming education, tracing programs means running a mental model that encompasses both the notional machine and the traced program [29]. The inherent impossibility to trace the execution of SQL queries limits the mental status representations that novices can have of the query elements and might support their 'black box' view of the DBMS. Solutions that have been proposed for assisting in the visualization of intermediate query results are the eSQL tool [14] as well as, more recently, concept maps of the evolution process of intermediate results of SQL statements [26].

Several of the misconceptions that we identified are specific to SQL and its instruction. From existing work, it is known that the application of templates to achieve a divide-and-conquer approach is effective in SQL instruction [23]. In our study, however, we observed students being confused or making errors when attempting to apply query templates that were not appropriate for the query formulation problem they were presented with. Another source of confusion were the semantics of SQL notation. This last cause of misconceptions relates to the internal consistency of the SQL syntax. There is ongoing discussion on this topic, with several aspects of the SQL syntax being considered counter intuitive [10, 16, 26]. In contrast, Taipalus attributes many of the most common SQL formulation errors they found previously [34] to ignorance [31]. Two examples of errors that they attribute to ignorance are *omitting a join* and *incorrect comparison operator, or value compared.* In our study we go beyond this perfunctory category and find evidence for more serious issues, such as complete confusion regarding the JOIN concept.

Limitations. We note several possible limitations of our study. First, the identified errors and misconceptions could be the result of the applied instruction strategies and teaching styles. The students that participated in our study were taught by one of at least three different teachers, but still the misconceptions that they hold and expressed might not be representative of another student population, taught by other instructors and in other institutions. However, our goal in this study was to understand misconceptions of novices independently of teaching style, and variations across the three participating subgroups do not significantly impact our findings. Second, the paper-and-pencil medium used during the interviews perhaps facilitated some syntax errors that might not have appeared while working on the computer. Nevertheless, students did use incorrect syntax regardless of the medium and hence the impact of the paper medium on our findings regarding syntax is quite limited. Third, the encoding and analysis of errors by the authors followed an existing error taxonomy [34] and hence inherits any limitations of this prior work. However, as categorizing errors was not the focus of our work, this has limited impact on our findings. Finally, two students reported difficulties solving the problems while thinking out loud. Whalley and Kasto [37] introduce a retrospection phase in their study of programming errors using audio playback. We follow a similar approach, but did not play back audio to students. Such an explicit audio playback might have been helpful for us to obtain further insights.

7 CONCLUDING REMARKS

In this paper we present a categorization of misconceptions for SQL learners. We base this categorization on a qualitative study in which we used a think-aloud methodology to understand the thought process of novices during query formulation problems. We found that misconceptions fall into four broad categories: misconceptions based in previous course knowledge, generalization-based misconceptions, language-based misconceptions, and misconceptions due to an incomplete or incorrect mental model.

Our findings provide a basis for further research into improving query language education. We close by indicating two natural avenues for future work. First, the prevalence of misconceptions and the extent to which misconceptions are context dependent (e.g., sensitivity to teaching style) should be investigated. Second, query language instructional approaches and materials should be revisited to understand their (in)adequacy in effectively addressing the misconceptions of novices.

REFERENCES

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

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

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

[4] Piraye Bayman and Richard E. Mayer. 1983. A Diagnosis of Beginning Programmers' Misconceptions of BASIC Programming Statements. *Commun. ACM* 26, 9 (Sept. 1983), 677–679. https://doi.org/10.1145/358172.358408

[5] Katinka Beker, Jasmine Kim, Martin Van Boekel, Paul van den Broek, and Panayiota Kendeou. 2019. Refutation texts enhance spontaneous transfer of knowledge. *Contemporary Educational Psychology* 56 (2019), 67–78. https://doi.org/10.1016/j.cedpsych.2018.11.004

[6] Erik Blair. 2015. A reflexive exploration of two qualitative data coding techniques. *Journal of Methods and Measurement in the Social Sciences* 6, 1 (2015), 14–29.

[7] Benedict Du Boulay. 1986. Some Difficulties of Learning to Program. *Journal of Educational Computing Research* 2, 1 (1986), 57–73. https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9

[8] 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.

[9] Michael Clancy. 2004. *Computer Science Education Research.* Taylor & Francis Group, Chapter Misconceptions and attitudes that interfere with learning to program, 85–100.

[10] Mohammad Dadashzadeh. 2003. A Simpler Approach to Set Comparison Queries in SQL. *Journal of Information Systems Education* 14, 4 (2003), 345–348. https://aisel.aisnet.org/jise/vol14/iss4/1

[11] Kathi Fisler. 2014. The Recurring Rainfall Problem. In *Proceedings of the Tenth Annual Conference on International Computing Education Research* (Glasgow, Scotland, United Kingdom) *(ICER '14).* Association for Computing Machinery, New York, NY, USA, 35–42. https://doi.org/10.1145/2632320.2632346

[12] John P. Smith III, Andrea A. diSessa, and Jeremy Roschelle. 1994. Misconceptions Reconceived: A Constructivist Analysis of Knowledge in Transition. *Journal of the Learning Sciences* 3, 2 (1994), 115–163. https://doi.org/10.1207/s15327809jls0302_1

[13] Lisa C. Kaczmarczyk, Elizabeth R. Petrick, J. Philip East, and Geoffrey L. Herman. 2010. Identifying Student Misconceptions of Programming. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (Milwaukee, Wisconsin, USA) *(SIGCSE '10).* Association for Computing Machinery, New York, NY, USA, 107–111. https://doi.org/10.1145/1734263.1734299

[14] R. Kearns, S. Shead, and A. Fekete. 1997. A Teaching System for SQL. In *Proceedings of the 2nd Australasian Conference on Computer Science Education* (The Univ. of Melbourne, Australia) *(ACSE '97).* Association for Computing Machinery, New York, NY, USA, 224–231. https://doi.org/10.1145/299359.299391

[15] L. Ma, J. Ferguson, M. Roper, and M. Wood. 2011. Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education* 21, 1 (2011), 57–80. https://doi.org/10.1080/08993408.2011.554722

[16] Victor M. Matos and Rebecca Grasser. 2002. A Simpler (and Better) SQL Approach to Relational Division. *Journal of Information Systems Education* 13, 2 (2002), 85–88. https://aisel.aisnet.org/jise/vol13/iss2/2

[17] D. McCall and M. Kölling. 2014. Meaningful categorisation of novice programmer errors. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings.* 1–8. https://doi.org/10.1109/FIE.2014.7044420

[18] Daphne Miedema. 2019. *Towards successful interaction between Humans and Databases.* MSc Thesis. Eindhoven University of Technology. https://pure.tue.nl/ws/portalfiles/portal/143091031/Miedema_D.pdf

[19] 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 '83.* ACM Press, New York, New York, USA, 161–165. https://doi.org/10.1145/800045.801602

[20] Roy D. Pea. 1986. Language-Independent Conceptual "Bugs" in Novice Programming. *Journal of Educational Computing Research* 2, 1 (1986), 25–36. https://doi.org/10.2190/689T-1R2A-X4W4-29J2

[21] 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.

[22] Ralph T. Putnam, D. Sleeman, Juliet A. Baxter, and Laiani K. Kuspa. 1986. A Summary of Misconceptions of High School Basic Programmers. *Journal of Educational Computing Research* 2, 4 (1986), 459–472. https://doi.org/10.2190/FGN9-DJ2F-86V8-3FAU

[23] Gang Qian. 2018. Teaching SQL: A Divide-and-Conquer Method for Writing Queries. *J. Comput. Sci. Coll.* 33, 4 (April 2018), 37–44.

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

[25] 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

[26] 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

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

[28] Juha Sorva. 2008. The Same but Different Students' Understandings of Primitive and Object Variables. In *Proceedings of the 8th International Conference on Computing Education Research* (Koli, Finland) *(Koli '08)*. Association for Computing Machinery, New York, NY, USA, 5–15. https://doi.org/10.1145/1595356.1595360

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

[30] Alaaeddin Swidan, Felienne Hermans, and Marileen Smit. 2018. Programming Misconceptions for School Students. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (Espoo, Finland) *(ICER '18)*. Association for Computing Machinery, New York, NY, USA, 151–159. https://doi.org/10.1145/3230977.3230995

[31] Toni Taipalus. 2020. Explaining Causes behind SQL Query Formulation Errors. *Proceedings - Frontiers in Education Conference, FIE* 2020-October (2020), 1–9. https://doi.org/10.1109/FIE44824.2020.9274114

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

[33] Toni Taipalus and Ville Seppänen. 2020. SQL Education: A Systematic Mapping Study and Future Research Agenda. *ACM Trans. Comput. Educ.* 20, 3, Article 20 (Aug. 2020), 33 pages. https://doi.org/10.1145/3398377

[34] Toni Taipalus and Mikko Siponen. 2018. Errors and Complications in SQL Query Formulation. *ACM Transactions on Computing Education* 18, 3 (2018), 29 pages.

[35] C Welty. 1985. *Correcting user errors in SQL.* Technical Report. 463–477 pages.

[36] Charles Welty and David W. Stemple. 1981. Human factors comparison of a procedural and a nonprocedural query language. *ACM Transactions on Database Systems* 6, 4 (dec 1981), 626–649.

[37] Jacqueline Whalley and Nadia Kasto. 2014. A qualitative think-aloud study of novice programmers' code writing strategies. *ITICSE 2014 - Proceedings of the 2014 Innovation and Technology in Computer Science Education Conference* (2014), 279–284. https://doi.org/10.1145/2591708.2591762

[38] Daniel Zingaro, Cynthia Taylor, Leo Porter, Michael Clancy, Cynthia Lee, Soohyun Nam Liao, and Kevin C. Webb. 2018. Identifying Student Difficulties with Basic Data Structures. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (Espoo, Finland) *(ICER '18)*. Association for Computing Machinery, New York, NY, USA, 169–177. https://doi.org/10.1145/3230977.3231005