On Learning SQL

On Learning SQL

Disentangling concepts in data systems education

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van
de rector magnificus prof. dr. S.K. Lenaerts,
voor een commissie aangewezen door het College
voor Promoties, in het openbaar te verdedigen op
vrijdag 12 januari 2024 om 16:00 uur

door

Daphne Eveline Miedema
geboren te Woerden, Nederland

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof. dr. M.G.J. van den Brand
1^{e} promotor:	prof. dr. G.H.L. Fletcher
2^{e} promotor:	dr. E. Aivaloglou
leden:	prof. dr. E. Ventura-Medina
	prof. dr. ir. F.F.J. Hermans (Vrije Universiteit Amsterdam)
	prof. A. Petersen (University of Toronto Mississauga)
	dr. S.S. Bhowmick (Nanyang Technological University)
	prof. dr. S. Scherzinger (Universität Passau)

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Colophon

SIKS Dissertation Series No. 2024-01

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Printed by: Ridderprint

Cover design: Daphne Miedema and Dennis Collaris

Cover: Picture from the Fairy Pools on Skye, Scotland. The winding path represents a learning journey that can be twisting and turning, but ends with mastery.

A catalogue record is available from the Eindhoven University of Technology Library with ISBN 978-90-386-5917-6, and an electronic version of this dissertation is available online at `https://research.tue.nl`.

Man, unlike any other thing organic or inorganic in the universe, grows beyond his work, walks up the stairs of his concepts, and emerges ahead of his accomplishments.

John Steinbeck

Contents

1

Introduction

1

1.1 Database Management Systems

The relational model was established in 1970 by Codd as a way to organize structured data. The model stores all data grouped together in a set of tables related to one another by means of keys: unique values present in both tables. The relational model allows for declarative access to data, which means that users only specify *what* they want, and leave it up to the system to decide *how* to get the data. Relational Database Management Systems, which use the relational model, have become the most popular data systems architecture [164].

Shortly after the model's inception, Chamberlin and Boyce introduced a query language for the model: The Structured Query Language, first called SEQUEL [36], later abbreviated as SQL [35]. Although it did not completely adhere to the relational model, the language quickly followed the model in becoming widespread. SQL still is a prevalent programming language. The 2022 IEEE Spectrum analysis of job openings in Computer Science shows that SQL is the most-requested language for developers [30]. As such, all Computer Science degrees should teach their students SQL to prepare them for their careers.

1.2 Computing Education Research (CER)

Computing Education Research is the field of study that focuses on understanding the phenomena and processes involved in teaching and learning computing. It has been around as long as SQL has: its flagship conference SIGCSE had its first iteration in 1970. The area is an interdisciplinary one, applying pedagogy and learning sciences to computer science. It considers all branches of computing education, from Kindergarten to vocational training and lifelong learning.

Currently, five primary conferences for CER work are SIGCSE (also called the Technical Symposium), ITiCSE, ICER, Koli Calling, and WiPSCE. Each of these has its own area of attention and as such the target audiences differ. For example, SIGCSE attracts many practitioners, WiPSCE is for primary and secondary school research, and ICER is focused on the development of theory and ground-breaking work. For example, the ICER website states:

> "We particularly encourage work that goes beyond the community's past focus on introductory programming courses in post-secondary education [...] We value new topics, new methods, new perspectives, and new ideas, just as much as more broadly accepted ones." [51]

The branch of research in the CER field that is of particular interest for this dissertation is that of programming education. Under this umbrella, researchers have published many works, of which the methodologies and findings might be transferable to SQL education research. Specifically, related topics include:

- exploring **factors that influence how easily a student will learn** a programming language, such as works on code comprehension [122, 159, 160, 189].
- characteristics that can help students **transfer their practice from one programming language to another** [155, 187, 188].
- the development of **practice and assessment materials** such as Parsons problems [52, 121, 205] and Concept Inventories [29, 56, 73, 133, 196].

- **learning trajectories**, such as Rich et al.'s meta-review on what are appropriate learning goals for programming in K-8 [149], and the work by Dasgupta et al. on remixing concepts [49].
- understanding the **barriers and difficulty in learning programming concepts**, including various works on misconceptions [144, 171, 204].
- **developing instruments** to support students learning to program, such as Hedy (a gradual programming language) [74], and PLTutor (for program tracing) [125].

1.3 Motivation

Currently, in many Computer Science bachelor curricula, a significant chunk of Data Systems courses are dedicated to Relational Databases and SQL. We should be teaching this, and teaching it well, to set our students up for success. To be able to teach our courses in the best possible way, we need both constructive alignment (making sure the classroom activities, assessment methods, and learning goals match [17]), as well as domain-specific insights. This is where CER comes in, with its insights into learning computing topics, development of tools, and methodological instruments. And although CER explicitly welcomes research on more advanced topics in computing degrees (such as Data Systems work), there are few contributions in this area. To be able to teach Data Systems courses well, research on student behavior, underlying theory, and good assignments is essential.

The benefits of such research can be seen in introductory computer science courses such as CS1. Over the past decades, CER conferences have published hundreds of works on CS1. A quick Google search for *CS1 course* gives close to 60000 results. There is work on factors that contribute to success and failure, diverse student populations, enrollment, self-efficacy, and many other dimensions. All of this background research has contributed to the development of several theory-backed curricula for CS1.

Computing education research specific to Data Systems courses so far has focused almost exclusively on exploring learners' errors in SQL. As such, the development of curricula depends on the authors of Data Systems books alone. Although these authors are Data Systems experts who understand their field well, they often do not have experience in educational research or pedagogy. Therefore, explanation methods and exercise types are not backed by theory and might not be optimal for learning. In addition, research has identified aspects of SQL that lead to high complexity: strict syntax rules, the cognitive load associated with translating the language, lack of correct and complete error messages, and interference of existing knowledge. Given the age and ubiquitousness of SQL, it is unlikely that changes to these aspects will be adopted, so education should also try to mitigate these problems. Finally, we have noticed in our workshop on Data Systems Education that each lecturer has their own preference for which topics to teach in an introductory course [5]. Different topics are valued differently by different lecturers.

In conclusion, there is a lot that teachers can gain from the integration of educational research methods into Data Systems Education. In this dissertation, I aim to tackle a focused subset of this work by exploring SQL education. As discussed above, research has shown that SQL is challenging to learn. As such, the benefits of theory-backed SQL education are significant.

1.4 Objective

The main research question for this dissertation is:

How can we support students in mastering challenging SQL concepts?

To answer this research question, we have worked on two streams of research. The first is to gain a deeper understanding of students' struggles. In the second stream, we consider ways in which we can support our students in Data Systems education. This dissertation is characterized by its exploratory nature, with all contributions having exploratory features to some extent.

Understanding students' struggles. Although SQL seems like a straightforward language to learn, with a syntax that is close to natural language, students do struggle with learning the language [3, 91, 118]. In the decades since SQL's inception, there has been research on many reasons why this is the case (see chapter 2 for more info on this). Unfortunately, not as much focus has been placed on the educational angle: how can we remedy the problem? In chapters 3 through 6, we will explore what might be some of the underlying causes of students' struggles. This first stream of research can be used to adapt our teaching and lessen the burden on the students.

Supporting students. Besides changing our teaching to focus on areas of struggle, we can also support our students with broader, course-wide changes. First, a course can become more manageable for students by offering them tools of support: peer review, discussion forums, quizzes, educational videos, or a better programming environment. Second, the course design itself can become more manageable. Changes may comprise the inclusion or exclusion of specific topics and contexts, as well as the way in which we offer these topics to our students. In chapters 7 and 8, we explore two such interventions.

1.5 Outline & Contributions

In this dissertation, I explore the aspects of SQL that students struggle with and analyze two interventions that could support students learning SQL.

Chapter 2 is the Background chapter, which provides the reader with information about the important concepts surrounding the research in this dissertation. The chapter describes research on recurring topics.

The main contributions of this dissertation can be found in Chapters 3 - 8. **Chapter 3** discusses our investigation into students' problem-solving process for query formulation in SQL. We examine the complexity of all intermediate attempts from four dimensions: correctness, execution order, edit distance, and query intricacy. We identify a student behavior type that we call self-inflicted complexity, in which cases students add unnecessary complexity to their queries during formulation and do not remove this even in case of errors.

In **Chapter 4** we take our analysis of problem-solving to the next level in a think-aloud study for query formulation. We interviewed 21 students, who answered approximately five questions each. Through our analysis of the students' thought processes and intermediate and final answers, we were able to create the first documented set of student misconceptions for

SQL. This set contains twelve misconceptions, grouped into four categories: misconceptions based on previous course knowledge, generalization-based misconceptions, language-based misconceptions, and misconceptions due to an incomplete or incorrect mental model.

Chapter 5 extends the work on misconceptions with another perspective: that of SQL experts. These teachers, researchers, and practitioners have a wealth of knowledge on SQL and its potential problems. We offered them a subset of the student errors that we found in the previous chapter, for which they shared many additional reasons why students might make these mistakes. This broader view gives us additional insights into why students may struggle. All of these reasons together can help us identify where our teaching material may be lacking and how to change our work to make learning more effective.

In **Chapter 6** we investigated the representativeness of previously identified misconceptions. As the studies in Chapters 3 and 4 were undertaken with only small sample sizes, we wanted to know whether they would also hold for larger, more diverse, student populations. We gathered misconceptions from the work in Chapter 4 and created multiple-choice questions that should evoke these misconceptions. These were then included in an online questionnaire, which was taken by approximately 300 students across the globe. We found that all misconceptions were held to some extent, with prevalence scores ranging from one to fifty-two percent. As such, we conclude that our misconceptions as identified previously, do indeed hold for a more diverse student population. This chapter wraps up the *Understanding students' struggles* part of this dissertation.

Chapter 7 is the first chapter in the *Supporting students* part of the dissertation. In this chapter, we focus on query formulation, which is challenging for students to do correctly. Therefore, we introduce a visual tool that supports students in this task. It is called SQLVis and shows a representation of the query structure as written by the student. The fact that SQLVis makes the relations between tables explicit helps the students understand whether they connected tables correctly. They can also reflect on whether what they wrote in SQL is what they meant to write. In our qualitative study, participants suggested that SQLVis could be useful for finding mistakes and for keeping track of their translation from natural language to SQL. The quantitative study showed that students using SQLVis have a higher proportion of correct answers, as well as a lower number of attempts.

In **Chapter 8** we discuss how to make Data Systems Education engaging for students. Unfortunately, Data Systems courses are typically seen as difficult, and not necessarily fun. Engagement may lead to lower drop-out rates and an increase in performance. To explore the students' perceptions of what an engaging database would be, we investigated their answers to a homework exercise where they had to design a relational database that they considered engaging for novices. We analyzed their reports from three dimensions: what is an engaging domain, what is an engaging number of relations and attributes, and what is an engaging amount of data. We identified six factors that students considered to make an engaging domain, five factors on engaging complexity, and three factors for engaging amounts of data.

Finally, in **Chapter 9** we summarize our contributions. We do this by exploring the overarching themes of Data Systems Education in this dissertation: problem-solving, misconceptions, and engagement. We also propose implications for practice, as well as research directions for the future.

	Qualitative Quantitative	Participant expertise	Number of participants	Type of contribution
Chapter 3		novice	104	problem solving
Chapter 4		novice	21	misconceptions
Chapter 5		expert	19	misconceptions
Chapter 6		novice	249	misconceptions
Chapter 7		intermediate / novice	5 / 43	problem solving
Chapter 8		intermediate	54	engagement

Figure 1.1: Chapter characteristics over the dimensions of research methods.

1.6 Overview of Research Methods

In this dissertation, we employ a variety of research methodologies. We identified three axes over which we have performed research: the type of work, the type of research methods, and the experience level of our participants. Figure 1.1 contains an overview of these dimensions.

As required for all work involving human subjects, all studies in this dissertation have been approved by the Ethical Review Board of Eindhoven University of Technology. This ensures that the studies are done with the utmost care for the privacy and safety of the participants and that the data is stored securely.

1.6.1 Type of output

Most of the chapters in this dissertation focus on gathering and **analysis** of education data. However, we also have two contributions that instead focus on **development** of tools for education.

The first is the work concerning SQLVis (chapter 7). In this chapter, we design, implement, deploy, and analyze a visual query representation. The system is built in D3.js, and packaged as a Python library to allow for fast and open-source distribution.

Our second development work is that of the Multiple-choice SQL Miconception Instrument, also called MSMI1 (chapter 6). The chapter discusses the development of an alternate type, namely that of educational material. In a five-step process, we developed a survey that can test whether students hold misconceptions. The five steps were: question formulation, question check by experts, pilot study, deployment, and question independence analysis.

1.6.2 **Type of research**

In our chapters focusing on data analysis for education, we applied various types of analysis methods. Most chapters have elements of both **qualitative** and **quantitative** analysis, while others specifically focus on one or the other.

In chapter 3, our work is **quantitative**. In this study, students in our introductory Database course received Jupyter notebooks that they could work on their homework in. The notebooks generated logs, which could then be uploaded to the Learning Management System. For this study, we analyzed students' problem-solving behavior by calculating various metrics. This exploratory study did not use multiple conditions, and as such we did not perform any statistical tests. However, our results are fully based on the metrics and analysis of plots based on these numbers.

Two chapters in this dissertation are mostly **qualitative**. The first is chapter 4, where we identified misconceptions that novices hold. To be able to identify these, we conducted semi-structured, think-aloud interviews, in which we asked students to solve query formulation problems of increasing difficulty. We then followed a data-driven approach to categorize both the errors and their underlying thought process. The second qualitative study is chapter 8, where we asked students in an advanced course to develop a database (schema and data) that would be engaging for novices. We then analyzed their reflection reports through directed content analysis: supported by existing literature we tried to find evidence of previously identified factors, but also coded new factors.

The other three chapters have elements of **both** types of analysis. First, chapter 5 uses a design inspired by the policy-Delphi, that we performed in two rounds. The goal of the study was to identify expert's hypotheses on student errors. In the first round, we presented our experts with student errors and asked them to generate as many causes as possible. We then aggregated all answers into a list of hypotheses, which we sent out to the same experts for voting. We asked them whether they thought the hypothesis was likely and why (not). We then created plots displaying our experts' votes to gain deeper insights into potential misconceptions and also analyzed their explanations.

Second, in chapter 6, we (developed and) deployed MSMI1, an instrument to identify SQL misconceptions. The questionnaire contains multiple-choice questions, which we analyzed for frequency. We also checked whether each pair of questions we developed was dependent, using a χ^2 test. Finally, the MSMI included space for students to reason about their answers. From our analysis of these reflections, we were able to identify an additional 10 misconceptions.

Finally, in chapter 7 we developed a visual query representation tool. In our evaluation process, we ran both a qualitative and a quantitative study. The qualitative study aimed to identify participants' general impressions of the system. In the quantitative part, we measured students' performance on homework in a between-subjects design (one group with access to SQLVis, one group without). We compared students' correctness, incorrectness, and number of errors between the two groups by means of t-tests. SQLVis seems to help students be more effective in query formulation.

1.6.3 Type of participants

We distinguish between three levels of participant expertise: **novice**, **intermediate**, and **expert**. All three levels have been investigated in this dissertation on their relation with SQL.

Our chapters involving **novices** are chapter 3, chapter 4, and chapter 6. In the first, we identified problem-solving behavior, in the latter two we investigated misconceptions. For both of these branches of research, insights into novices' experiences are most useful for teaching practice, as it is easiest to subvert novices' thought patterns.

We worked with **intermediate** level students (those currently taking or having taken a graduate level database course) in chapter 8 and the design phase of chapter 7. In both of these, we were designing interventions for novices. We felt that novices themselves might not have a good enough grasp on SQL to be able to judge the interventions. Intermediate-level students, on the other hand, are able to strongly relate to novices while also having a solid view of what might (not) be interesting and/or helpful. We later tested SQLVis on novices in chapter 7, but without asking them for feedback on the design.

Finally, we worked with **experts** in chapter 5. We invited these experts because we felt they could offer a complimentary perspective. Our experts all worked with SQL often, either as practitioners, researchers, or teachers. Therefore, they have a good view of common mistakes in their own context. This helped us identify new viewpoints on novices' mistakes.

1.7 Publications

Publications in scientific conference proceedings and journals:

- **Miedema, D., Fletcher, G., and Aivaloglou, E.** So many brackets! An analysis of how SQL learners (mis)manage complexity during query formulation. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension* (Virtual event, 2022), pp. 122–132 [114].
  This publication serves as core material for Chapter 3.

- **Miedema, D., Aivaloglou, E., and Fletcher, G.** Identifying SQL Misconceptions of Novices: Findings from a Think-Aloud Study. In *Proceedings of the 17th ACM Conference on International Computing Education Research* (Virtual Event, USA, 2021), pp. 355–367. This publication was reprinted in ACM Inroads as "Worth reading from ICER 2021" [111] and received a **Honorable mention**.
  This publication serves as core material for Chapter 4.

- **Miedema, D., Fletcher, G., and Aivaloglou, E.** Expert Perspectives on Student Errors in SQL. *ACM Transactions on Computing Education (TOCE) 23*, 1 (2022), 1–28 [113].
  This publication serves as core material for Chapter 5.

- **Miedema, D., Liut, M., Fletcher, G., and Aivaloglou, E.** "There is no ambiguity on what to return": Investigating the Prevalence of SQL Misconceptions. In *23rd Koli Calling International Conference on Computing Education Research* (Koli, Finland, 2023) [116] and
  Miedema, D., Liut, M., Fletcher, G., and Aivaloglou, E. MSMI1: Towards a Validated SQL Misconceptions Instrument. In *Proceedings of the 2023 ACM Conference on International Computing Education Research V.2* (Chicago, USA, 2023) [115].
  These publications serve as core material for Chapter 6.

- **Miedema, D., and Fletcher, G.** SQLVis: Visual Query Representations for Supporting SQL Learners. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (Virtual Event, USA, 2021), IEEE Computer Society, pp. 1–9 [112].
  This publication serves as core material for Chapter 7.

- **Miedema, D., Taipalus, T., and Aivaloglou, E.** Students' Perceptions on Engaging Database Domains and Structures. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto, Canada, 2023), pp. 122–128 [117] and
  Taipalus, T., Miedema, D., and Aivaloglou, E. Engaging Databases for Data Systems Education. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland, 2023), pp. 334–340 [177]. This paper was nominated for a **Best Paper Award**.
  These publications serve as core material for Chapter 8.

Publications to which I contributed during my PhD but that are not included in the dissertation:

- **Miedema, D., Aivaloglou, E., and Fletcher, G.** Exploring the prevalence of SQL misconceptions: a study design. In *Proceedings of the 21st Koli Calling International Conference on Computing Education Research* (Virtual event, 2021), pp. 1–3 [110].

- **Aivaloglou, E., Fletcher, G., and Miedema, D.** DataEd'22-1st International Workshop on Data Systems Education: Bridging Education Practice with Education Research. In *Proceedings of the 2022 International Conference on Management of Data* (2022), pp. 2556–2557 [6].

- **Aivaloglou, E., Fletcher, G., Liut, M., and Miedema, D.** Report on the First International Workshop on Data Systems Education (DataEd'22). *ACM SIGMOD Record 51*, 4 (2023), 49–53 [5].

- **Collaris, D., Weerts, H. J., Miedema, D., van Wijk, J. J., and Pechenizkiy, M.** Characterizing Data Scientists' Mental Models of Local Feature Importance. In *Nordic Human-Computer Interaction Conference* (Aarhus, Denmark, 2022), pp. 1–12 [44].

- **Aivaloglou, E., Fletcher, G., and Miedema, D.** DataEd'23 - 2nd International Workshop on Data Systems Education: Bridging Education Practice with Education Research. In *SIGMOD '23: Companion of the 2023 International Conference on Management of Data* (Seattle, USA, 2023), pp. 313–314 [7].

- **Miedema, D.** Toward a Fundamental Understanding of SQL Education. In *Proceedings of the 2023 ACM Conference on International Computing Education Research V.2* (Chicago, USA, 2023) [109].

- **Odynchuk, A., Miedema, D., and Fletcher, G.** SQLVis+: Restoring Visual Query Representations in Case of Syntax Errors. Under review [128].

- **Orlov, D., Miedema, D., and Yakovets, N.** A Cup of ChaiSQL: Benefits of Type Checking for SQL. Under review [130].

2

Background

In this Chapter, background information is provided on the concepts discussed in this dissertation. There are three recurring themes throughout this dissertation: cognition during programming, errors in SQL, and misconceptions. Existing work on these three themes is gathered in this chapter and can be used as a reference whenever there is a need for more background.

2.1 Cognitive processes during programming and querying

Research on human aspects of Structured Query Language (SQL) started shortly after its inception in 1974. For example, Chamberlin himself mentions that the definition of Sequel 2 was in part based on the experiences of early users [35]. In 1981, Phyllis Reisner ran a meta-review on query language studies, including SQL, with the goal of illustrating behavioral research on query languages [147]. Welty and Stemple in that year compared SQL and TABLET on their associated human factors [194]. Back then, the JOIN keyword did not yet exist, and instead joining tables was done implicitly through the WHERE clause. They found that students struggle with this syntax, and write

> "We believe that the underlying SQL semantic model is too diffuse to allow either easy learning or good retention. This is further exacerbated by use of the totally different syntactic construct of nesting for chaining, an operation which semantically overlaps joining." [194, p. 642]

A later study by Greene et al. on cognitive skills and their relation to SQL performance showed that reading skills, reasoning ability, spatial memory, and integrative processing ability were four cognitive skills significantly correlated with query formulation performance.

More generally, confusion during programming and querying can be caused by the cognitive processes involved in the retrieval of information from long-term memory, short-term memory, and working memory [75, p. 6]. Limitations in the attention and memory resources for cognitive processing have been identified as possible causes of programming errors [131]. Researchers have found that they also play a role in the complexity of writing SQL queries [146, 157].

Regarding **long-term memory**, there might be interference from learners relying on what they already know. As learners have typically worked with other programming languages before, knowledge transfer from another language to SQL may lead to mistakes [146]. Differences in the notations and their use for the definitions of variables and arithmetic operators are especially prone to transfer and thus misconceptions [153]. For example, see the differences in the usage of = and == for assignment and comparison. Another problem with previous programming language knowledge is that learners typically have experience with imperative languages, in which they can split the problem up into parts and debug it. In a declarative language such as SQL, novices cannot split up the problem in steps as much but have to apply set-based thinking [152].

Natural Language may also interfere with long-term memory. Learners might be (incorrectly) transferring concepts from natural language to SQL [22, 40]. Keywords borrowed from the English language can have ambiguous or different meanings [22, 141], such as the word **and**, which is a conjunction in natural language but a Boolean operator in coding. Another example is that novice programmers believed that an **if** statement was continuously actively waiting for the Boolean condition to be true, as used in natural language [135].

On the other hand, long-term memory involvement is required, as SQL syntax seems simple but is sometimes counter-intuitive. Thus, query formulation requires an in-depth understanding of SQL keywords and their corresponding data transformations [157]. The strict syntax also leads to low expressive ease and errors [38].

During SQL query formulation, **short-term memory** needs to store and process information about the database schema [119, 129, 173] and the WHERE clause conditions [4] to be able to accurately translate the question into a query. A highly complex database structure also leads to unnecessary complications and difficulties in refactoring queries [173].

Finally, the **working memory** is affected by a few aspects of SQL. Shin writes about how learners must integrate all correlated data in memory to understand query execution [157]. This leads to high cognitive load as, in SQL, relations between tables are implicit, and need to be reconstructed from foreign keys. Then there is the lack of syntactic locality in SQL [1], which means that a small change in logical syntax (for example ∃ to ∀) requires a largely different query. This is aggravated by the fact that the Database Management System (DBMS) only responds to syntax errors, making it difficult for users to find semantic errors [119], as there is often no indication of how to locate these errors. Plus, for SQL, error messages are regularly incomplete and their suggestions incorrect [176], also making it harder to recover from errors. Lastly, the DBMSs only report one error at a time, which gives an incomplete perspective on the correctness of queries.

2.2 Errors in SQL

In the previous section, we investigated the cognitive processes that are behind many struggles during query formulation. These struggles lead to frequent errors, which have been examined by many researchers. Early work focused on ease-of-use of various keywords in query formulation [148], and misspellings and omissions [162]. Newer works have focused on organizing all observed error types over different categories.

Three error categories are commonly reported: syntax errors, semantic errors, and complications.

Syntax errors are errors in the query that prevent compilation of the query. DBMSs will always return an error message in case of a syntax error. Such errors may include misplacement of punctuation, misspelled keywords or relation names, or SQL keywords in an incorrect order [2]. Due to SQL's strict syntax rules, syntax errors are common.

Semantic errors are those queries that return an unintended result, but no error message. This means there is a mismatch between what the user wants the query to encode, and what they have written down. A very common example among beginners is to create a Cartesian product without adding JOIN conditions [4, 26]. The result of such a query is much larger than intended. The DBMS will not catch semantic errors, as it does not know what the user was intending and the query as written is valid.

Semantic errors as defined above are sometimes called logical errors too [28, 180]. Taipalus et al. distinguish between logical errors and their interpretation of semantic errors, which they define as a query that is incorrect regardless of what the question was. Such queries may have a constant output, a join on a column that never matches, or an inconsistent expression that results in an empty result table such as in Query 2.1. Other researchers do not distinguish between semantic and logical errors and call both categories semantic errors [26, 162].

```
SELECT name
FROM staff
WHERE city = 'Boston'
AND city = 'Chicago'
```

Query 2.1: Example of a query with an inconsistent expression [162].

```
SELECT id
FROM staff
WHERE salary > 1500
OR salary > 1700
```

Query 2.2: Example of a query with an implied condition [26].

Finally, queries with **Complications** are syntactically and semantically correct but have low readability. Some examples include the use of DISTINCT on a primary key, a duplicate output column, or implied conditions (see Query 2.2).

This line of research has led to the identification of various SQL concepts that students seem to struggle with, such as:

- (self-)JOINs [2, 4, 37, 91, 118, 172, 178]
- GROUP BY [4, 91, 119, 147, 172, 178, 194]
- HAVING [2, 4]
- aggregation [4, 37, 91, 172]
- correlated subqueries [3, 4, 118, 138, 178]

One important research direction is to examine error significance; which errors are worse for students to hold? Taipalus and Perälä studied persistent errors, those that are present in the student's final answer [178]. They found that logical errors and complications had the highest incidence, and that syntax and semantic errors occurred least in final attempts.

Much work is still to be done in charting all SQL errors. For example, Taipalus and Seppänen indicate the need for further study of SQL including advanced SQL features such as recursion and Common Table Expressions, advanced SQL concepts such as assertions and triggers, and realistic research settings [179].

Another important research direction is that of error message design. As we discussed in section 2.1, the fact that the DBMS only reports one syntax error at a time is not ideal for users. In general, research and design of error messages for SQL have been understudied [176]. Two papers comparing error messages between DBMSs show that there is a wide variety in the error messages and the perceived usefulness thereof [174, 176]. The content of the messages is not always accurate, may be incomplete, or is too general, and thus does not help students fix their queries [175]. Therefore, Taipalus and Grahn suggest a usability-based framework for improving SQL error messages, suggesting nine guidelines for design, and propose sixteen error messages for the most common queries [175]. It remains to be seen whether industry will adopt these guidelines.

On the other hand, significant progress has been made in tool development for supporting students. There exist various tools for supporting query formulation through different media: [67, 70, 126]. Tools were also developed to give students feedback on their queries [50, 94, 127]. Other tools identify code smells to help students improve or repair their queries

[104, 124], or use visual elements to remove some of the students' mental load during query formulation [33, 59, 100, 112].

2.3 Misconceptions

Between the struggles in section 2.1 and their erroneous results in section 2.2, there is a translation mechanism of conceptions. Conceptions can be referred to as belief systems about the workings of the world. If a person holds an incorrect belief, we call this belief a misconception. In education, understanding misconceptions and their triggers is essential. If we want to help our students learn to use SQL more effectively, we need to know the nature of these misconceptions that result in difficulties or erroneous code being written. Most of the work on misconceptions in coding has been done on programming languages. Until recently, there were no qualitative works on misconceptions for SQL.

Misconceptions have been considered "faulty extensions of productive prior knowledge" that should be identified and refined rather than replaced [163, p. 152]. In programming, students' prior knowledge, especially from natural language and mathematics, has been found to be transferred and cause misconceptions [40, 144]. According to DuBoulay, an important source of misconceptions are properties that students assume of the machine that executes their code [22]. Furthermore, Pea identified the assumption that "there is a hidden mind somewhere in the programming language that has intelligent, interpretive powers" [135, p. 2].

Misconceptions on introductory programming have been categorized as either difficulty in syntactic knowledge, conceptual knowledge, or strategic knowledge, which refers to knowledge about planning, writing, and debugging programs [144]. Research on programming misconceptions has mainly focused on the understanding of concepts in imperative and object-oriented languages [102, 165] – an extensive list of misconceptions is presented by Sorva [166].

Several works have attributed misconceptions to the misapplication of prior knowledge of mathematics, especially algebraic notation. Bayman found that they cause confusion on variable assignment statements [12] and Sorva identified them as affecting the understanding of execution sequences [165].

One of the few works on SQL misconceptions is written by Smelcer, who developed a model to explain known errors [162]. 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 dissertation.

Finally, Taipalus [172] discusses causes that could be behind the persistent SQL errors explored in [178]. They map the errors to four of the cognitive explanations as introduced by Smelcer [162].

In this dissertation, we extend the work on misconceptions in CER drastically.

Part 1 - Understanding student struggles

3

Measuring Complexity during Query Formulation

In our journey to support students in learning complex SQL concepts, we first aim to learn more about how students approach query formulation. Their problem-solving process can help us understand more deeply where their first issues occur, and how we might design interventions to help them. In this chapter, we study intermediate attempts on query formulation problems, with a focus on complexity. We utilize four perspectives on complexity: 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. We call this behavior self-inflicted query complexity.

The contents of this chapter have previously appeared in **Miedema, D., Fletcher, G., and Aivaloglou, E.** So many brackets! An analysis of how SQL learners (mis)manage complexity during query formulation. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension* (Virtual event, 2022), pp. 122–132 [114].

3.1 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, 26, 146]. The most comprehensive investigation to date has been undertaken by Taipalus, Siponen, and Vartiainen [180], who distinguish between syntactic, semantic, and logical errors and complications. Various researchers have examined reasons why novices make many mistakes [1, 38, 111, 119, 152]. One important factor is the student's existing knowledge, which influences their approach [152] and can lead to misconceptions [111].

The effects of program complexity on comprehension and cognitive load have also been widely studied [54]. Specifically for SQL, high complexity is problematic. As an illustration, see Query 3.1 (and its corresponding question in Table 3.1), 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)))
```

Query 3.1: 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, 119, 129], as well as the relation between complexity and accuracy play a role. Siau et al. found an inverse relation between query complexity and accuracy [158] and Ion finds that complex SQL queries (defined through element count) are more prone to semantic errors [84]. Taipalus found that the complexity of the database structure leads to unnecessary complications and difficulties in refactoring queries [173].

	Exercise text	SQL concepts
Exe. 1	List all the product ids that were bought by at most two different customers.	self-join °, does not exist ♠
Exe. 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 ♠
Ex. 3	List the ids of the customers that made a purchase at every store.	correlated subquery °, does not exist ♠
Ex. 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 ♠
Ex. 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 ♠
Ex. 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 ♠

Table 3.1: The exercises to be answered by the participants. Concepts with a ° are used by Ahadi et al. [3] and Taipalus et al. [180], concepts with a ♠ were introduced by Taipalus et al. [180].

Second, from a technical perspective, complexity is also the subject of much research. For example, progression in the capabilities of AI with regard to Natural Language processing has led to renewed interest in 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 [156]. Maintainability and readability are also affected by specific complexity aspects of SQL queries, such as the number of tables and the degree of nesting within the query [136].

In this chapter, 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.

3.2 Related work

Our work in this chapter is related to and builds on studies by Taipalus et al. [178] and Migler et al. [118].

Taipalus and Perälä, define a category of problems called *complications* [178]. Complications are unnecessary elements that make it difficult to understand a query, although the query is

	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 3.2: Correctness and attempts for all exercises.

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 [178].

Migler and Dekhtyar 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 [118]. They argue that we should evaluate student learning by measures beyond success and errors.

Similar to Migler and Dekhtyar [118], we analyze both intermediate and final attempts at 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 [178] 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 [118].

Taipalus and Perälä [178] 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 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.

3.3 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.

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 3.3: The database schema. Underlined attributes indicate the primary keys.

Ethics. The participants were not financially compensated for participation in the study. The study design and data collection were approved by the Ethical Review Board of Eindhoven University of Technology.

3.3.1 Data Collection and Filtering

We provided all participating students with a Jupyter notebook for a graded assignment on SQL. The notebook contained cells that enabled the students to run Python code. The assignment consisted of six exercises testing different concepts (Table 3.1) to be evaluated over the schema in Table 3.3. 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 3.1.

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 answers 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.

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

3.3.2 **Data processing**

To answer our research question we explore query complexity from four 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 the correctness of all queries in our collection. We used four categories: correct, semantic error, syntax error, and time-out [26, 66, 197]. 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. Previous work has shown that tracing students' work can give us insights into students' strategies [72]. The traces we identify here also reflect the student's self-regulation process [8].

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.

Edit distance. Edit distance analysis shows how different each newly defined query is from the one before. Do students keep struggling with 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[99] in which we count the differences in *words* instead of the differences in *characters*. This is similar to the differences in SQL structure used by Yang et al.[197]. 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.

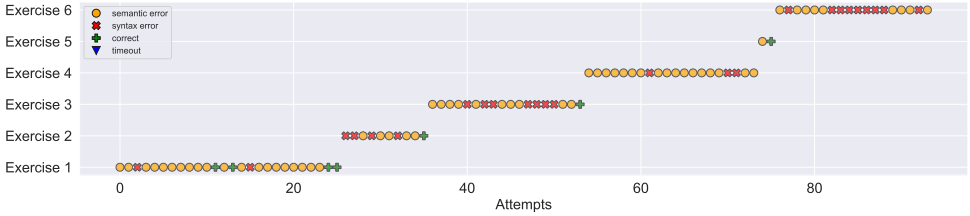


Figure 3.1: An execution order plot for Participant 31. This participant approaches the problems in order.

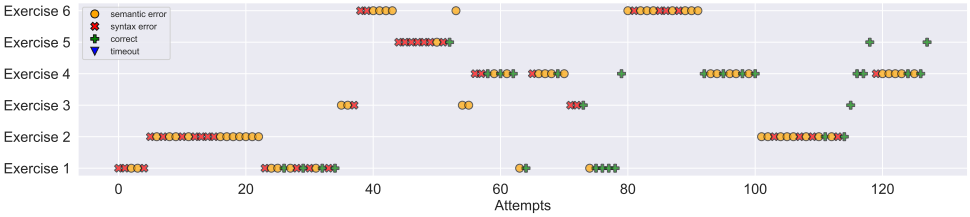


Figure 3.2: An execution order plot for Participant 99. This participant switches between exercises often.

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 compared 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 [136].

3.4 Results

3.4.1 Correctness and attempts

Our analysis of overall correctness and attempts shows that on average, some exercises were more difficult than others. For the full statistics, see Table 3.2. 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. Students struggled to write 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.

3.4.2 Execution order

Our analysis of 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 3.1)

and 99 (Figure 3.2). 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 of 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 Query 3.1), 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 the 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 counterproductive, and in this case, might have even led 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.

3.4.3 **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 an edit distance of 0, we are left with 7,757 attempts. In Figure 3.3 we see that the majority of attempts (49.5 percent or 3837 attempts) have a change ratio less than 0.1.

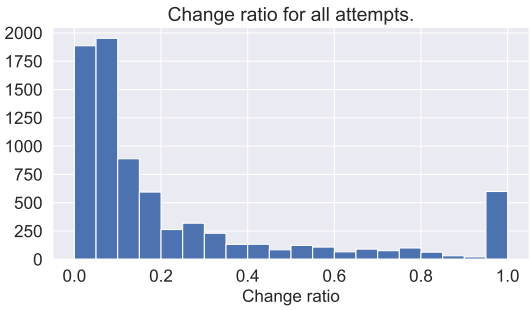


Figure 3.3: Change ratio < 0.1 is most common, meaning that participants typically make small changes.

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 either a large addition to the query or a large removal. For examples, see the queries of Participant 15 in section 3.6. Once we examine the contents of the query logs in combination with these plots, we find some interesting behavior.

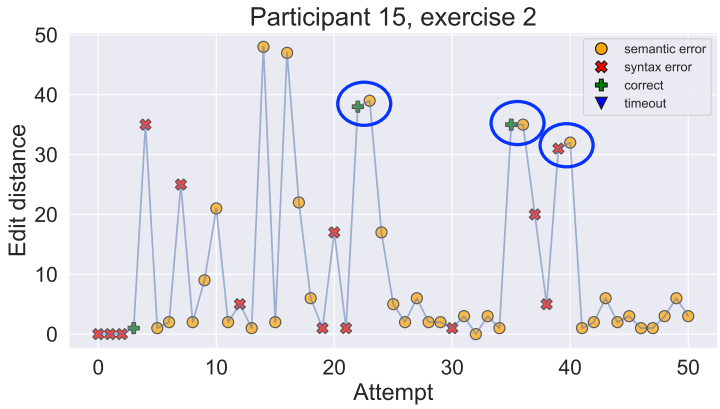


Figure 3.4: Participant 15 makes large changes throughout their attempts with differences of up to 39 words.

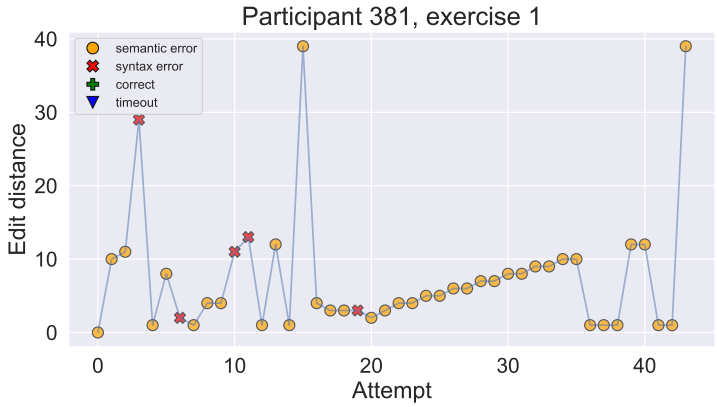


Figure 3.5: Participant 381 gradually makes edits.

First of all, let us consider Participant 15 in Figure 3.4. 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 student 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 section 3.6. So, a set of similar edit distances reflects a temporary change of approach.

The edit distance plot for Participant 381 (Figure 3.5) shows gradually increasing edit distances between attempts 20 and 35. Inspection of the query logs shows that this participant tried to solve the problem unconventionally: manually checking the purchase IDs that should be included, and then hard-coding them in a query. For an excerpt from the process, see Query 3.2. These switches result in a slightly increasing edit distance, which is a bit misleading as there is no real progress happening.

	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	-	-

Table 3.4: Number of query elements. The right-most column counts the attempts with more query elements than the correct answer (in the second to right column).


```
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)
```

Query 3.2: Participant 381, exercise 1, 9 consecutive attempts at solving the problem in a round-about way.

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 the sunk cost fallacy.

3.4.4 Query intricacy

Finally, our analysis of query intricacy highlights extreme values for the measures introduced in subsection 3.3.2. The scores for query elements can be found in Table 3.4 and scores on brackets in Table 3.5 and Table 3.6. 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 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 3.5 and Table 3.6 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 attempts with high intricacy scores, we find overly complex SQL text as a result of students' continuous struggles. For an example, see Query 3.3, in which a student queried with 124 elements and eight sets of brackets.

	0	1	2	3	4	5	6	7	8	9	10
Exercise 1	692	401	175	39	22	3	0	0	0	0	0
Exercise 2	958	349	207	137	92	9	9	5	3	2	0
Exercise 3	298	413	293	252	140	57	32	4	0	0	0
Exercise 4	251	649	390	255	61	34	24	2	5	0	0
Exercise 5	136	454	383	194	39	17	12	1	0	0	3
Exercise 6	299	281	262	230	125	53	44	16	16	0	1

Table 3.5: Attempts per bracket set count per question.

	0	1	2	3	4	5
Exercise 1	692	467	139	4	0	0
Exercise 2	959	572	169	65	6	0
Exercise 3	298	650	512	24	5	0
Exercise 4	251	1013	360	38	9	0
Exercise 5	136	507	529	47	19	1
Exercise 6	300	556	430	39	2	0

Table 3.6: Attempts per bracket nesting depth per question.

Brackets do not always indicate subqueries. They are also used for view definition (Query 3.3) 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.

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.

```
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)
```

Query 3.3: A highly intricate attempt by Participant 410, demonstrating self-inflicted complexity. Indentation and newlines were added for readability.

3.5 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, 26, 118, 139, 146, 162]. Towards understanding the reasons for these errors, our analysis of intermediate attempts gives insight into 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 a 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 [178]. 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 produced errors to more complex queries that still produced 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 [54], McCabe's Cyclomatic Complexity [92, 120] and Halstead's metric [120]. 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 [92] -the problem decomposition issue becomes apparent in our work on SQL complexity too. Algorithmic complexity was identified as a problem for novice programmers [151], which expert programmers are able to tackle by adopting effective strategies. For SQL, those strategies might relate to the use of query templates [142].

Compared to the work by Yang et al. [197] 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 a 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 [197]. This does not match our findings. This may be in part because 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 [173], 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.

3.5.1 **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 decom-

posing the problem [144]. Decomposition can be supported by tracing programs: running a mental model that encompasses both the notional machine and the traced program [166]. 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 [91] and SQLVis [112]), as well as concept maps of the evolution process of intermediate results of SQL statements [157].

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 [96] by applying pattern-oriented instruction [86]. In SQL instruction, recent work in this direction has proposed the application of templates to divide-and-conquer query formulation problems [142].

Toward helping novices internalize SQL syntax, patterns, and templates, two techniques that could help are retrieval practice and active elaboration [76, p. 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 [199]. However, the application of those techniques to SQL instruction remains to be investigated.

3.5.2 **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 were 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 at only six exercises, versus the larger sets in other papers. Although we cover many of the concepts that students find most difficult [178], 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 arise in this case. 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 of 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. To capture the 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 Table 3.2 may look worse than they would have been in case of feedback on correctness. To mitigate this, we also report the percentage of participants with at least one correct answer.

3.6 **Appendix: Examples 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

Table 3.7: 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 attempts 12 and 13, to approach B in attempts 14 and 15, and finally back to approach A in attempt 16.

Attempt	Query	Correctness	Edit distance

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

Attempt	Query	Correctness	Edit distance
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
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

4

Identifying Students' SQL Misconceptions

Our findings in the previous chapter quantify the struggles that students have with writing SQL. However, what chapter 3 does not tell us is why this is the case. Therefore, our next focus is on identifying the underlying causes of student struggle: misconceptions. In this chapter, we study misconceptions through semi-structured interviews and the think-aloud method. In our results, we start by exploring the errors made during query formulation. Then, we expand on this through the exploration of the think-aloud results, to determine the participants' misconceptions. In total, we identify twelve misconceptions grouped into four themes: misconceptions based in previous course knowledge, generalization-based misconceptions, language-based misconceptions, and misconceptions due to an incomplete or incorrect mental model.

The contents of this chapter have previously appeared in **Miedema, D., Aivaloglou, E., and Fletcher, G.** Identifying SQL Misconceptions of Novices: Findings from a Think-Aloud Study. In *Proceedings of the 17th ACM Conference on International Computing Education Research* (Virtual Event, USA, 2021), pp. 355–367. This publication was reprinted in ACM Inroads as "Worth reading from ICER 2021" [111].

4.1 Introduction

Databases and the Structured Query Language (SQL) are core topics in Computer Science curricula in higher education [179]. The syntax of SQL is relatively simple in comparison to most programming languages, but SQL is not trivial to learn. Many works have reported on errors in the SQL queries written by students and novices, identifying categories such as syntax errors, logical errors, semantic errors, and complications [4, 26, 139, 172, 178, 180]. In the research investigating 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 toward 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 [135], difficulties of learning to program, and errors based on the misapplication of analogies [22]. 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 [40, 144]. One of the causes of difficulties and misconceptions that has been widely reported on is the misapplication of students' prior knowledge [144, 163]. 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 [22, 40]. Prior exposure to other programming languages has also been found to cause misconceptions [153], 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 five underlying causes: working memory overload, absence of retrieval cues, procedural fixedness, incorrect procedural knowledge, and misperception [162]. Recently, errors in the SQL queries submitted by students were mapped to these five causes by Taipalus [172], without however utilizing qualitative input from the students.

In this chapter, 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, [90, 165]) 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.

4.2 Related work

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 [172] discusses causes that could be behind the persistent errors explored in [178]. They map the errors to four cognitive explanations as introduced by Smelcer [162]. Unfortunately, this mapping is speculative [172], as they only use the plain text query as submitted, without qualitative discussion with students.

With the knowledge of SQL errors and programming exercises that we presented in chapter 2 we aim to further the work in SQL education by identifying learners' difficulties and misconceptions. In contrast with the work by Taipalus [172], 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.

4.3 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 [55]. 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.

Ethics. The participants were not financially compensated for participation in the study. The study design and data collection were approved by the Ethical Review Board of Eindhoven University of Technology.

4.3.1 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 4.1.

Interview questions The question pool for the think-aloud interview is available in Table 4.3, with the schema for the database represented in Table 4.2. 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 4.3 we also mention *covered concepts*. These are based on the seven types of SQL queries defined by Ahadi et al. [3].

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?	6/21 (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 4.1: 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 4.2: The database schema. Underlined attributes together form the primary key for each table.

The starting question was chosen based on the self-reported SQL score. For participants with a 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.

4.3.2 **Participants**

For our participant pool, we were looking for students who had recently learned SQL. For recruitment, we reached out to database course instructors the authors knew. Interested instructors forwarded our interview invitation to their students.

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 were Dutch, 15 of the interviews were conducted in Dutch.

	Question type	Question	Covered concepts	Tried by
1	Execution	What is the result of Query 4.1? What is the natural language equivalent?		11
2	Formulation	List all IDs&names of customers living in Eindhoven.	Simple query	14
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	20
5	Advanced formulation	Find the names of all inventory items that have a higher unit price than Bananas.	Singlesubquery	14
6	Advanced formulation	Return a list of the number of stores per city	Group By	5
7	Advanced formulation	Return the stores table ordered alphabetically on city.	Order By	4
8	Execution	What is the result of Query 4.2? What is the natural language equivalent?		5
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.	Highcomplexity	2

Table 4.3: The pool of questions available for the interview.

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

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

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

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


```
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 4.1: 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 4.2: Second execution trace question.

For education level, our participants fell into 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, although five had work experience in programming languages.

Table 4.1 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.

4.3.3 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

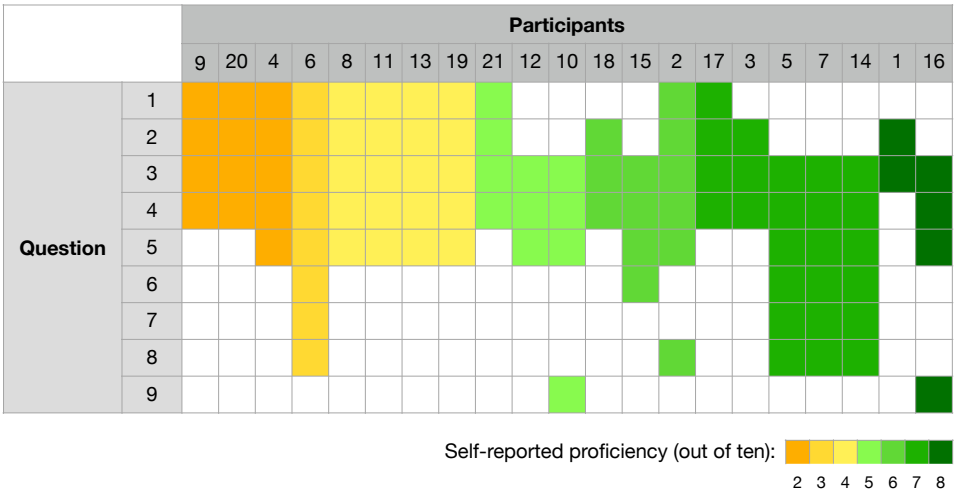


Figure 4.2: Summary view of attempts per participant, ordered by self-reported proficiency. The figure shows that self-reported proficiency was not necessarily a good indicator for performance.

ideas behind the study and 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.

We introduced the database schema and some example data (see Figure 4.1) 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 4.3. 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 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. An overview of which participant attempted which questions, and their self-reported SQL poficiency can be found in Figure 4.2.

4.3.4 Data processing

For data analysis, we used the notes of the participants on their intermediate and final answers to the interview questions in Table 4.3 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 [26, 180]. The error categorization is presented in section 4.4.

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 [18]. 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.

4.4 Identified SQL errors

After the interviews, the first intermediate step towards exploring students' misconceptions was 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 chapter. 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.

The presentation of the errors below is the 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 4.5.

4.4.1 Syntax issues

Syntax errors are a common error category, present in various papers on SQL errors [2, 139, 180, 193, 194]. 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 showed 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 commas 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.

4.4.2 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' [4]. 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 [180], and Welty et al.'s minor operand error [194]. Errors regarding attributes and their types are categorized by Welty as complex errors [193]. This category does not occur in Brass et al.'s list of semantic errors [26].

4.4.3 Incorrect or missing keyword

Other researchers classify these errors as syntax errors [26, 180], 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 a missing or extra operator in their semantics category [139]. 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.

4.4.4 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 [180]. Similarly, it includes some errors listed by Welty and Stemple: minor language error, minor substance error, and major substance error [194]. Incorrect results also include some errors reported by Presler-Marshall et al.: wrong values in where, wrong ordering, wrong operator in WHERE [139]. It does not occur in Brass et al.'s list of semantic errors [26].

4.4.5 Issues with the database schema

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

As we discussed in the related work chapter of this dissertation, 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 about 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.

4.4.6 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 [180]. From Presler-Marshall et al., alias problems are captured under column reference error [139]. This category is not included in Brass et al.'s list of semantic errors [26].

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

Query 4.3: Participant 16, question 2, an example of a contracted query.

4.4.7 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 4.3).

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 [26]) to be difficult. The main aggregation issue we found is where GROUP BY is required, which resulted in several incorrect answers.

4.4.8 **Complications**

Complication is a common category supported by previous literature [26, 180] 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, and C as aliases. This significantly reduces readability, and for these participants, we saw some swapped aliases, where incorrect sets of aliases and columns 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 a foreign key in another table. This resulted in extra JOINs that were not required.

4.5 **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.

4.5.1 **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 learned 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.

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 valid operations 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.

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 4.4: 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 the timing of if-statements being 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 4.4, 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 4.14), 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.

Databases.

Typically, students following a course on Databases are taught Relational Algebra first, with SQL taught afterward. 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 4.5: Participant 10, question 5.

Believing DISTINCT takes the first item of a list. For question 5 in Table 4.3, there was a follow-up question to gather more insights into 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 4.2), 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 4.5. 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 subsection 4.5.1, 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.

4.5.2 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.

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 4.6: 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 4.3, *Return the number of stores per city*, one participant counted the same column they grouped on, as can be seen in Query 4.6. 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.

Misunderstanding SQL syntax and its internal (in)consistency.

The consistency category is similar to the templates category but instead considers SQL syntax elements that students have difficulty with because they need to be 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 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 confusion 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 4.7: 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 4.7 and Query 4.8.

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

Query 4.8: Participant 4, question 5, alias behind the attribute in SELECT.

```
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 4.9: Participant 19, question 5, using a semicolon in the subquery.

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 4.9. 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.

4.5.3 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 subsection 4.5.1 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 4.10). 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 4.10: 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 [129, 162]. 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 4.10, but the latter may indicate a language-based misconception. One such example is shown in Query 4.11, 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 4.3) actually did not mention sort, but instead used "ordered alphabetically".

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

Query 4.11: Participant 14, question 7, participant used SORT instead of ORDER.

```
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 4.12: Participant 20, question 4, counting when this is not part of the question.

Usage of a non-native language. Finally, during the study we noticed some language issues. Many non-native English-speaking participants translated the problems into 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 into their native language, which changed the implications of the question. In the end, some of our translating participants (such as Participant 20, Query 4.12) arrived at the conclusion that they needed to count something, and thus applied the COUNT keyword.

4.5.4 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 4.13: 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 4.13). 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 [108] instead of trying to find the actual mistake and fixing it. This shows a gap in the knowledge of the students.

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

Query 4.14: Participant 18, question 3, an attempt at a self-join without a second table present.

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.

Thinking of the DBMS as a black box. The self-join error, described in subsection 4.5.1 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 4.3, the interviewer hinted to 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 4.14.

Another participant did not understand in which cases they needed to add JOIN conditions. Query 4.15 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 4.10. 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 discussed 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 to 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 4.15). 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 4.15: Participant 9, question 4, missing JOIN condition in Cartesian product.

4.5.5 **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 there. For example, for problem 5 in Table 4.3, 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 4.3, some participants noted that the 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.

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 4.3, 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 4.16: 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 4.16). 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 4.3, 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, some participants 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 4.17: Participant 4 using a subquery instead of a join.

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

Participant 4 decided to use a subquery for problem 4 in Table 4.3. 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 4.8 and Query 4.17 using the IN keyword. This is a cross-over of a 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 4.8. 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 of 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.

4.6 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 [40, 144]. 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 [163]. Several approaches have been proposed for addressing misconceptions in introductory programming [144], which could apply to SQL instruction. One of the approaches that has been lately demonstrated to be effective in facilitating the transfer of revised knowledge is refutation texts, which explicitly state and refute misconceptions [14].

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 4.8). 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." [91, 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." [178, 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." [3, 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 [144]. 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 [166]. The inherent impossibility of tracing 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 [91] as well as, more recently, concept maps of the evolution process of intermediate results of SQL statements [157].

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 [142]. 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 was the semantics of SQL notation. This last cause of misconceptions relates to the internal consistency of the SQL syntax. There is an ongoing discussion on this topic,

with several aspects of the SQL syntax being considered counterintuitive [48, 105, 157]. In contrast, Taipalus attributes many of the most common SQL formulation errors they found previously [180] to ignorance [172]. 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 [180] and hence inherits any limitations of this prior work. However, as categorizing errors was not the focus of our work, this had a limited impact on our findings. Finally, two students reported difficulties solving the problems while thinking out loud. Whalley and Kasto [195] introduce a retrospection phase in their study of programming errors using audio playback. We followed 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.

5

Experts' Opinions on Errors' Underlying Causes

With our findings in the previous chapter, we gain deeper insights into the gaps in SQL instruction that may lead to misconceptions. However, the setup of the study only allows for a limited sample of participants. Therefore, we aimed to identify more underlying causes by asking another group of participants: experts. Our expert group contained researchers, practitioners, and educators, who we believed could provide us with complementary insights given their experience with SQL. In a Policy Delphi-inspired study we first prompted the participants to generate explanations for student errors, then aggregated these and had the experts vote on the likelihood of each explanation. Our participants generated four to eight hypotheses for each error in the study, providing us with a wider interpretation of SQL misconceptions.

The contents of this chapter have previously appeared in **Miedema, D., Fletcher, G., and Aivaloglou, E.** Expert Perspectives on Student Errors in SQL. *ACM Transactions on Computing Education (TOCE) 23*, 1 (2022), 1–28 [113].

5.1 Introduction

The Structured Query Language (SQL) is ubiquitous, both in practice and in education. It is an established language, introduced in 1970 for querying relational databases [43] and has been an ISO standard since 1987 [85]. Research in Database Education has shown that SQL is difficult for learners, as indicated by the high prevalence of errors in SQL queries. Various researchers have studied these errors; one of the first such studies was done by Reisner [147], while more recent work has provided a more extensive understanding of errors [2, 4, 118, 180].

Only recently have researchers started to explore the underlying reasons for these SQL errors: misconceptions. In total, there are four types of conceptions, and all can be applied in instructional approaches to help students learn: correct, incomplete, incorrect, and misconceptions [103]. Margulieux et al. propose *multiple conceptions theory*, which suggests that "learners develop better conceptual knowledge when they are guided to compare multiple conceptions of a concept during instruction" [103, p. 194]. For this reason, it is important to explore all existing conceptions on SQL. Since all types of conceptions for SQL are underexposed, much work needs to still be done.

In the previous chapter, we employed a think-aloud protocol to identify student misconceptions on SQL resulted in fourteen misconceptions in four categories: misconceptions based in previous course knowledge, generalization-based misconceptions, misconceptions based in language, and misconceptions due to an incomplete or incorrect mental model. Building on these findings, we aim to explore the misconceptions of novices from a different viewpoint: that of experts. SQL experts can be educators, researchers, or practitioners. Each of these groups has a distinct view of SQL errors and their underlying causes, which strengthens the content validity of the study. Furthermore, as experts have worked with SQL significantly longer than novices, they should have correct and highly detailed mental models. This, in combination with their experience of teaching SQL, can give us unique insights into underlying causes of errors.

Research goals We aim to explore the underlying causes for student errors from the perspective of SQL experts. The method through which we do this is a derivative of the Delphi technique [41]. We selected eleven errors of different categories identified by Miedema et al. [111], and formed a panel of nineteen SQL experts from diverse backgrounds. We then presented these errors to our panel and asked them to provide their hypotheses as to why students might make these errors. We aggregated their answers to find all distinct hypotheses and set up a second questionnaire in which we asked all experts to vote on the likelihood of each hypothesis posed by them and their colleagues. In particular, we aim to answer the following two research questions.

Q1 How do experts interpret the underlying causes of student errors in SQL?

Q2 What are the differences in the perspectives of students and experts on errors in SQL?

Answers to these questions provide insights complementary to that of the student perspective, towards the design of effective educational approaches addressing and correcting the misconceptions underlying the SQL errors made by students.

The remainder of the chapter is structured as follows: we first introduce related work to our study in section 5.2. Then, we introduce the errors that this chapter is analyzing, by putting them in the context of existing literature in section 5.3. We present our research method in section 5.4, and our results in section 5.5. Finally, we discuss our findings, their validity, and limitations in section 5.6.

5.2 Related work

5.2.1 Cognitive resource use

Various authors have studied the cognitive causes of programming errors by novices. Limitations in the attention and memory resources for cognitive processing were identified as possible causes of programming errors [131]. Exploring the cognitive processing differences between expert programmers and novices, Bateson et al. found that experts make more use of semantic memory and high-level plan knowledge to direct their programming activities [11]. More recently, Yen et al. found that experts and novices have similar thinking processes in syntax debugging, but different processes in semantic debugging and logical structuring [200]. Examining the causes of different types of errors, Ko and Myers developed a model of programming errors in event-based systems that ties specific errors to their cognitive causes, which can be knowledge, strategic, or attentional, and can relate to specification, implementation, and debugging activities [95].

5.2.2 Teachers' perceptions on student errors

Related work in computer science education has examined the programming errors students make from the point of view of educators. Hristova et al. surveyed computer science professors about commonly made Java programming errors and identified the ones that are perceived as the most frequent and harder to find and fix for the students [80]. In a later study, Brown and Altadmri examined whether educators can make an accurate estimate and form a consensus on the frequency and time to fix 18 Java errors [27]. Comparing survey results with a dataset of compilation events, they found that educators' estimates did not agree with one another or with the actual student data and suggested that a different level of discourse may be required, as "educators may still be accurate about the cause of mistakes and a student's conceptions of the mistakes, but just not about the frequency and time-to-fix of such mistakes." [27, p. 19]

Examining teachers' understanding of the causes of student errors or their misconceptions is particularly important because it is a major factor in teachers' pedagogical content knowledge, affecting planning, conducting instruction, and assessment [132]. Teachers should be able to identify student misconceptions and have an understanding of the context in which they are created [79]. One of the studies that have focused on programming misconceptions was made by Qian et al., who surveyed teachers about 37 misconceptions of high school students on Python [143]. The study examined their perceived frequency, importance in learning, and teachers' confidence in addressing them, and found that misconceptions may be difficult for teachers to detect if they are latent, i.e., not immediately leading to error messages.

5.2.3 The Delphi protocol

The Delphi protocol is also referred to in the literature as the Delphi technique, the Delphi method, or just the Delphi. It was introduced to optimize the methods of interaction with experts [41]. When compared to discussing with several experts back to back, or through round-table discussions, the Delphi protocol has the benefit of experts evaluating each others' viewpoints, as well as the advantage that no loud voice can dominate the discussion [41]. Herman adds: "[E]xperts remain anonymous during the [Delphi] so that they are influenced by the logic of the arguments rather than by the reputations of other experts." [73, p. 50]

In the context of educational research, the Delphi protocol has been used as a starting point for the development of Concept Inventories (CIs) [64, 73, 169]. For CIs, typically researchers start by asking what concepts should be taught in a certain course. Then, for each concept that occurs in the list more than once, they ask their experts to rate them based on importance and student understanding. Specifically in the Computer Science education domain, Delphi has been applied in the works by Goldman et al. [63] and Kaczmarczyk et al. [90] on memory models and data assignment, the work by Herman on logic [73], the work by Wittie et al. on CS2 [196], and the work by Farghally et al. on algorithm analysis [56].

Instead of focusing on concepts, as the studies working towards a CI do, in our Delphi setup we aim to gather hypotheses. Experts vote on each others' hypotheses to gain more insight into what the more likely hypotheses are. This method is a variation of the Policy Delphi, which, in contrast to conventional and real-time Delphi, asks experts to present all options and evidence for considerations [41]. This type of Delphi method is not necessarily about making decisions but is more about being informed. The idea behind Policy Delphi is to identify differing opinions instead of reaching consensus [201].

5.3 Error Context

In our study on misconceptions, we present our participants with 11 errors for their consideration. All of these errors have been identified and described in related literature. The errors are of varying severity and specificity. For example, some are syntax errors and some are complications, i.e., errors that would still lead to a correct results table [180]. Some errors are specific and some are broader and represent general issues, such as concerns about the understanding of GROUP BY, JOIN, or subqueries. Some of these may not have been covered literally but instead can be considered part of a general error category.

In our deliberation below, we do not discuss the work by Miedema et al. [111], as that is the source of these errors. The literature context for each error is summarized in Table 5.3.

5.3.1 Using ≠ instead of != or <>

This error regards a non-standard operator used in SQL, one that is not recognized as correct. It is a syntax error, as parsing ≠ will lead to an error. This error is mentioned by Presler-Marshall et al. as a *broken operator* [139], and by Taipalus et al. as *SYN-6 Common syntax error: nonstandard operators* (36) [180]. In later work, Taipalus et al. highlight that errors of type SYN-6 are among the most persistent errors made by students [178]. Ahadi et al. specifically focus on SQL syntax errors in [2]. In their discussion of PostgreSQL error codes,

one code that occurs often is that for *operator does not exist*. Although Ahadi et al. do not explicitly mention the error we describe here, it does fit the aforementioned operator category.

5.3.2 AVG in WHERE instead of HAVING

The foundation for this type of error was laid by Ahadi et al. who identified student difficulties in all queries involving HAVING [3]. In a later paper, they write that "we noticed that the condition that is supposed to appear in the HAVING clause was often mistakenly written in the WHERE clause" [2, p. 275], which is the generalized variant of our error. Taipalus et al. identify two categories in which aggregation is misplaced: *SYN-4 Illegal aggregate function placement* (14) and *LOG-4 Expression error* (69) [180]. AVG in WHERE is a syntax error, making SYN-4 our best fit.

5.3.3 JOIN in all tables that one needs the primary keys of

This mistake can be categorized as a complication: it does not lead to an incorrect result table but does increase the computational complexity. Brass et al. introduced this concept of complication, and more specifically introduced the mistake of *unnecessary join* [26]. In the work of Taipalus et al., this mistake is most closely related to *complication 84 - unnecessary join* [180]. Another related error that they identify is the *LOG-2 JOIN error* (59), as their example shows the inclusion of an extra table. In later work, Taipalus et al. conclude that both complications and LOG errors are common [178]. The persistence of LOG-2 errors is unknown, but complications are typically corrected, according to them.

5.3.4 Missing second table for self-join

Again, Ahadi et al. laid the groundwork to show that students struggle with self-joins: they found that 76% of students made (unidentified) mistakes with the self-join [3]. Although the missing second table for self-join was not mentioned specifically, Ahadi et al. do indicate that students did not understand that they needed a self-join [2]. Presler-Marshall et al. capture this error with their categories of *incomplete query* and *table reference error* [139]. In the categorization of Taipalus et al., this error is covered in *SEM-3 Missing JOIN* (48) and *LOG-2 JOIN error* (62). SEM-3 is among the most frequent and persistent errors, and LOG-2 is frequent but with unknown persistence [178]. Taipalus also mapped errors to causes. For these two errors, the causes are the absence of cue and lack of knowledge of the students [172].

5.3.5 Lack of understanding of when to apply GROUP BY

GROUP BY is another concept that students struggle with: these struggles are mentioned in almost all SQL error-related papers. Already in 1977, Reisner found many problems of the 'major error' class in GROUP BY questions [146]. This high error rate was also found by Ahadi et al. [2, 3]. Furthermore, Ahadi et al. identified the errors *irrelevantly used GROUP BY clause* and *including extra columns in GROUP BY clause* [2].

Brass and Goldberg identified five types of errors with GROUP BY: unnecessary GROUP BY in EXISTS subquery, GROUP BY with singleton groups, GROUP BY with only a single group, Unnecessary GROUP BY attribute, and GROUP BY can be replaced with DISTINCT [26]. All five of these reflect a lack of understanding of how GROUP BY works, and when to apply it properly.

5.3.6 Missing parts of keywords

Various examples of this error are available in the literature. Ahadi et al. found that sometimes students would forget the ON in JOIN ON [2]. Presler-Marshall et al. group this error under miscellaneous errors, with the example of a student writing BY instead of GROUP BY [139]. Taipalus classifies this error as *SYN-6 Common syntax error* (36) [180]. Their later work confirms SYN-6 errors to be among the most persistent [178].

5.3.7 Extra condition when not required

This mistake can also be classified as a complication: the extra condition does not change the result table, but does make the query longer. Ahadi et al. include this mistake under the category *irrelevant condition in the WHERE clause* [2]. Brass and Goldberg group this under *implied, tautological or inconsistent sub-condition* [26]. In the work of Taipalus et al., this mistake is related to both *LOG-4 Expression error* (68) and complication 82 [180]. As mentioned before, LOG errors have been shown to be common, and this error again is a persistent one. Taipalus maps error 68 to ignorance of the students [172].

5.3.8 Ambiguous column name because of missing alias

This error is mentioned explicitly in various other works, either as ambiguous column [2], column reference error [139] or *SYN-1 Ambiguous Database Object* (2) [180].

5.3.9 Using comma instead of AND in the WHERE clause

Although this error is not mentioned in literature about the WHERE clause, several works do refer to similar errors. Reisner identifies a category of minor errors, which includes errors in spelling and punctuation [146]. Presler-Marshall et al. also identify punctuation errors, specifically for commas: they coin the errors *extra commas* and *missing commas* [139].

5.3.10 Thinking DISTINCT will take the first item of a list

This error is perhaps described in a non-optimal way. When searching through the literature, we found this error described in a more explanatory manner: subquery term that might return more than one tuple, by Brass and Goldberg [26]. The error of the student is that there was a subquery that may return more than one result, and this needs to be fixed in some way. The student then suggested DISTINCT, which does not work. It is noteworthy that we could only find one paper that mentions this error, as this seems to be an error that may occur often for novices.

5.3.11 Lack of understanding of subqueries

Subqueries are another element of SQL that students seem to struggle with. Ahadi et al. find that "Students were not able to identify the skill required to answer [questions requiring a simple subquery], or they did not have the skill required to construct such a query." [2, p. 275] and that "the identification by students of the need for a correlated subquery was the most common problem." [2, p. 275].

From the perspective of subqueries as complications, Brass and Goldberg define the error *IN/EXISTS condition can be replaced by comparison* [26], and Taipalus et al. repeat this error under complication 90 [180].

5.4 Method

5.4.1 Research Design Overview

Our goal for this study was to capture expert hypotheses on SQL errors made by novices. To this end, we presented novice errors to our expert participants and obtained their perspectives on them.

The study consisted of two rounds of online questionnaires, implemented through Google Forms for convenient distribution. The first questionnaire contained eleven SQL errors examined in the study by Miedema et al. [111], introduced in section 5.3. For each of these eleven problems, we asked the participants to formulate one or more hypotheses as to what the underlying causes could be for this problem.

For the second round, we collected all answers for each of the questions in Round 1 and aggregated them to produce lists of hypotheses. In the resulting questionnaire, we asked the participants to vote on the likelihood of each of these hypotheses. Furthermore, participants were able to comment on the questions and hypotheses of others. From the resulting qualitative input, we can draw conclusions on expert views on underlying causes for SQL errors made by novices, answering Q1. For deeper insights, we then mapped each hypothesis to causes introduced in literature [111, 162, 172]. The mapping was done in collaboration by two of the authors, where any cases of disagreement were discussed. This led to the confirmation of existing mappings and the development of some new categories:

- Previous course knowledge [111]
- Generalizations [111]
- Language [111]
- Mental model [111]
- Lack of practice
- Sloppiness
- Not an error
- Working memory overload [162, 172]
- Absence of retrieval cue [162, 172]
- Procedural fixedness [162, 172]
- Ignorance [162, 172]
- Shown examples were too limited
- Avoidance (database anxiety)

One additional category by Smelcer is that of Misperceptions. However, we exclude that here as the point was to find misconceptions (misperceptions), and as such all hypotheses could potentially be mapped to this category.

The new categories of *lack of practice*, *sloppiness* and *not an error* were recurring themes in the participants' open answers. These can be seen as underlying causes for errors, but do not have a knowledge component, and thus do not map to our earlier misconception work.

The new category *Shown examples are too limited* originated in the work of chapter 4, where we hypothesized that various errors were due to a lack of exposure. This category, as the new category *Avoidance*, were also recurring themes in the qualitative input and did not map to any of the existing categories. Therefore, the authors agreed to include these as additional categories.

Toward answering Q2, we extracted the misconceptions for each error from Miedema et al. [111] and compared them with the list of hypotheses generated from the above-mentioned process. We examined whether the two contained similar concepts or reasoning, to draw conclusions on the expert versus student perspective.

5.4.2 **Participants**

Recruitment Process

All participants were invited through an email, detailing the current state of research and the study. It included details on the type and content of the questionnaires, as well as an estimate for the amount of time each questionnaire would take.

The invitees then let us know via return email whether they would like to participate and were sent a link to the questionnaire. Both questionnaires were open for two weeks, with an invitation sent at the start of the first week, and a reminder in the second week.

We aimed for 15-30 participants in this study, as we believed that this number would be appropriate to saturate the different types of hypotheses our participants would propose. The participants' differing backgrounds concerning working experience, culture, and interests made saturation more likely.

Ethics. The participants were not financially compensated for participation in the study. The study design and data collection were approved by the Ethical Review Board of Eindhoven University of Technology.

Participant Characteristics

Our participants were experts in SQL education (either in tertiary education or mentoring and training in industry), use, and research. The experts were recruited from the authors' networks of researchers and educators and were invited personally with an introductory email. Those who elected to participate received the links to the questionnaires by email.

In total, we invited 27 people. Of the 27, 17 people work for (15 different) universities, 1 at a teaching college, and 9 in industry. They work in nine different countries (in alphabetical

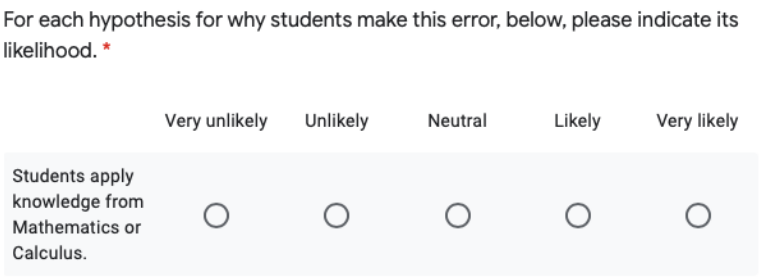


Figure 5.1: An example of a hypothesis to vote on.

order): Belgium, Chile, France, Germany, Netherlands, Poland, Singapore, the United Kingdom, and the United States. This geographic, cultural, and institutional spread increases the content validity and generalizability of the study. From this set of participants, we received 18 submissions to our first questionnaire, and 19 submissions to the follow-up.

To obtain deeper insights into population characteristics, we asked participants to identify their relationship with SQL. They could choose one or more relations out of the following five: educator, researcher, user, developer/architect, and consultant. Most of our participants classified themselves as educator, researcher, or both. An overview of participant characteristics is presented in Table 5.1.

	Round 1	Round 2
Educator	12	16
Researcher	9	7
Practitioner	7	8
User	4	6
Developer/ Architect	6	4
Consultant	4	2

Table 5.1: Participant profiles. Each participant belongs to one or more categories.

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 5.2: The database schema used for the questions in section 5.5. Underlined attributes together form the primary key for each table.

5.4.3 **Data Collection**

In the first round, we presented the participants with the eleven errors (see the first column of Table 5.3) along with contextual information on them, such as the query formulation question and an example SQL query demonstrating the error. A text field was provided in which the participants could provide their hypotheses on the cause of the error. The questionnaire was kicked off with an example, such that the participants had some ideas about our expectations. The questionnaire also had a field at the end for any further notes that the participant might like to make.

The next phase included the aggregation of hypotheses from the first round. For this process, the first author selected all groups of 18 hypotheses (one hypothesis per participant, one group per error). They then went through each group from the first submission to the last and summarized the core of each hypothesis if it had not been introduced before. Some participants wrote down their ideas in one-sentence hypotheses, whereas others wrote down elaborate reasons. After the first author had made a pass, the second and third authors compared the full list of answers to the aggregated list and checked whether all participants' ideas were included. Each error was left with four to nine hypotheses.

In the second round, the participants could then vote on the aggregated hypotheses on a five-point Likert scale, as can be seen in Figure 5.1. Each set of hypotheses also had space where our participants could elaborate on their answers if they wished to do so.

As we discussed in subsection 5.2.3, the Policy Delphi does not aim for consensus but for gathering opinions. It was therefore not necessary to establish consensus criteria to evaluate the hypotheses over.

5.5 Results

In this section, we aim to answer the research questions raised in section 5.1:

Q1 How do experts interpret the underlying causes of student errors in SQL?

Q2 What are the differences in the perspectives of students and experts on errors in SQL?

Before we dive into the participants' hypotheses per error, we first introduce our results with a basic overview. In Table 5.3, we present each error, its reference in existing literature, the number of proposed hypotheses by our participants, the most popular expert hypothesis, and the proposed student hypothesis from Miedema et al. [111].

In the following subsections, we cover each of these errors and hypotheses in turn to answer Research Questions 1 and 2. To give the reader the appropriate context, we also present the elements our participants received, to support the understanding of the error: (1) the question the students tried to answer, (2) a (partial) query containing the error, and (3) context on the error. The database schema that the questions concern can be found in Table 5.2. The material as presented in each section, is exactly the same as the presentation to our participants.

The participants voted on each of the aggregated hypotheses in part two through a five-point Likert scale. We visualize this by means of stacked bar charts with a diverging color scale, where the width of each segment corresponds to the proportion of answers. Each category of votes is shown in a different color (likely hypotheses in blue, unlikely hypotheses in orange), with the neutral answers centered on the midline of the plot. Below each hypothesis, a grey badge indicates the corresponding category as introduced in subsection 5.4.1.

The participants have been assigned random numbers and are referred to as Participant 1 to Participant 19.

Error	Similar error mentioned in:	Num of hypotheses	Most popular expert hypothesis	Student misconception according to Miedema et al. [111]
Using ≠ instead of != or <>	[2, 139, 180]	6	Students are familiar with ≠ and find it more intuitive than <> and !=	Interference of previous course knowledge from either Mathematics or Relational Algebra
AVG in WHERE instead of HAVING	[3, 180]	7	Students associate the application of a filter with WHERE	Misunderstanding SQL syntax and its internal (in)consistency
JOIN in all tables that you need the primary keys of	[26, 180]	7	Students have created a template query/ pattern and apply it without understanding	Applying an incorrect query template
Missing second table for self-join	[2, 3, 139, 172, 180]	8	Students do not think of a second access as a separate instance of a table	Misunderstanding the scope of elements in a query, or thinking of the DBMS as a black box.
Lack of understanding of when to apply GROUP BY	[2–4, 26, 146]	7	Students do not understand the implication of grouping: there are no rows anymore and thus one cannot project on them	Misunderstanding SQL syntax and its internal (in)consistency
Missing parts of keywords	[2, 139, 180]	5	Students forget the correct syntax because most other SQL keywords are single-valued	Remembering only the core part of a keyword (as this captures the 'main' meaning)
Extra condition when not required	[2, 26, 172, 180]	5	Students do not realize that being a different customer follows from the different cities (as cID is the primary key and thus unique)	Lacking knowledge on primary keys
Ambiguous column name because of missing alias	[2, 139, 180]	4	Student knew which pID they mean and feel it is self-explanatory	No specific misconception for this error is provided. Instead, they attribute this to a lack of knowledge or experience.
Using comma instead of AND in WHERE clause	[139, 146]	6	Student was sloppy in writing the query	Misunderstanding SQL syntax and its internal (in)consistency
Thinking DISTINCT will take the first item of a list	[26]	4	Students mix the semantics of 'a single' and 'the first'	Believing DISTINCT takes the first item of a list (because of being presented with very selective examples of DISTINCT queries)
Lack of understanding of subqueries	[2, 26, 180]	7	Students prefer a loop-style query, as you would do in programming	No specific misconception for this error is provided. Instead, they attribute this to a lack of knowledge or experience

Table 5.3: This table presents an overview of the results, containing all 11 errors explored in this chapter, the number of hypotheses that came out of Delphi round one, and the most popular hypotheses that came out of Delphi round two. For each error, we also compare to the misconception underlying this error from the student perspective, taken from [111].

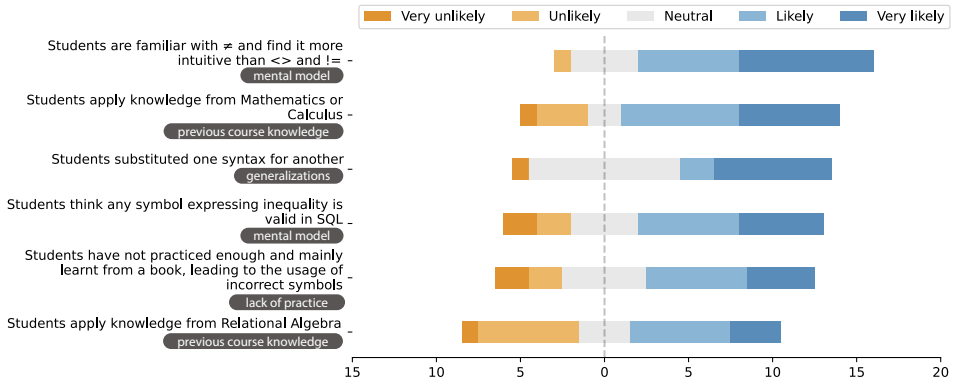


Figure 5.2: Expert hypotheses for *Using ≠ instead of != or <>*

5.5.1 Using ≠ instead of != or <>

Findings for this error are displayed in Figure 5.2. Overall, the trend in these expert hypotheses is that ≠ is a common and intuitive symbol. Students may have used it for several reasons. Our Delphi participants are sympathetic, which is reflected in their hypotheses:

> "[..] in general, I sympathize with this one, ≠ is just the way it is written in all normal math on paper (and is also the way it should be in sql if it wasn't for the missing ≠-key on the keyboard...(and the lack of support for ≠ in ASCII))" - Participant 1

> "It is a common symbol that will have been used in other courses, and the meaning is actually correct. Plus the <> and != in SQL are not that intuitive in their form in connection with their meaning." - Participant 2

The fact that ≠ is common in other areas of study is also reflected in the mapping of hypotheses: two of them can be classified as *previous course knowledge* and one under *generalizations*. Furthermore, one hypothesis can be mapped to *lack of practice*, and two to *mental model*.

Regarding Q2, the students and experts have similar perspectives. The fact that students have learned of the ≠ symbol months to years before learning SQL, means it is more accessible in memory than more newly introduced symbols such as <> or !=.

5.5.2 AVG in WHERE instead of HAVING

Provided problem statement

Question:
Find the names of store-chains that on average sell products in quantities of more than 4.
Error context: This placement of AVG does not lead to a correct result as the average should be taken over the grouped attribute.

Student (partial) answer:

```
GROUP BY t.sID
FROM transaction as t
WHERE AVG(t.quantity) > 4;
```

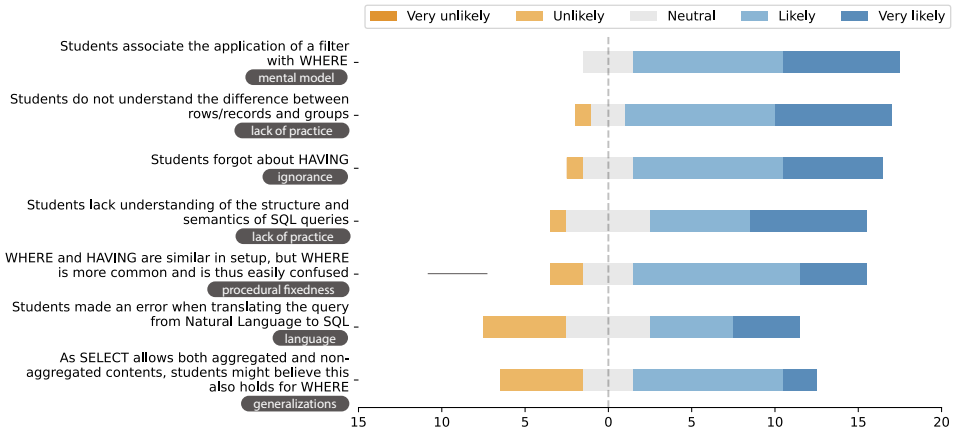


Figure 5.3: Expert hypotheses for *AVG in WHERE instead of HAVING*

Findings for this error are displayed in Figure 5.3. The hypotheses cover a very wide spectrum of underlying issues. Again, we see a reflection of the 'this is more common' approach, but we also see considerations of the concept of groups, the concept of aggregates, and the concept of filter. Furthermore, translation could be an issue, as well as plain forgetfulness. Interestingly enough, there seems to be no correlation between the background of our participants, and the type of hypothesis they present for this error.

Selection at different stages was central to many participants' answers, for example:

> "WHERE and HAVING both do some form of selection, but just at a different stage. The type of statements (boolean combinations) also look very similar. So it is not so strange to confuse them. [...]" - Participant 2

The difference between processing individual rows and groups is the main distinction between WHERE and HAVING. This processing difference -combined with having highly similar syntax and functionality- might be confusing, so Participant 18 suggests renaming HAVING to "WHERE GROUP HAS" in the SQL syntax.

Also related to this is the 'flow' or the order of processing of a query, and how students may not be familiar with it. Not knowing the order of processing and the implicit restrictions that come with it, may lead to this error. Participant 8 illustrated this by writing:

> "[...] student did not understand that WHERE clause is evaluated before GROUP BY is applied" - Participant 8

As mentioned above, these hypotheses cover a wide spectrum of issues. As a result, most of them fit into different cause categories: only two were mapped to the same category of *lack of practice*. The other categories of causes for this error are *language*, *procedural fixedness*, *ignorance*, *mental model* and *generalizations*.

The top hypothesis of the experts and the students reflect similar thoughts. The concept of filter seems to be mainly associated with WHERE by both student and expert users. If there

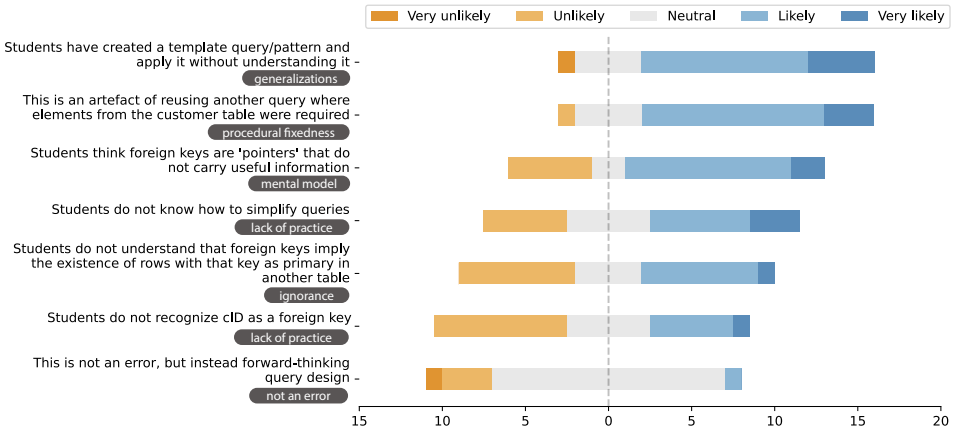


Figure 5.4: Expert hypotheses for *JOIN in all tables that you need the primary keys of*

is a condition, students will write this in the WHERE as that is consistent with what they know. The consistency of this misconception is also reflected in the hypotheses listed 5th and 7th in Figure 5.3.

5

5.5.3 JOIN in all tables that you need the primary keys of

Provided problem statement

Question:
List all customer IDs, dates and quantities of transactions containing products named Apples.
Error context:
Here the participant joined in the customer table for cID, although cID is already available in transaction as a foreign key.

Student answer:

```
SELECT c.cID, t.date, t.quantity
FROM customer as c,
    transaction as t,
    product as p
WHERE t.pID = p.pID
AND c.cID = t.cID
AND p.pName = "Apples"
```

Findings for this error are displayed in Figure 5.4. For the first time, we see some hypotheses that skew to neutral or even disagreement.

Most of the expert answers in round one discussed ways in which students approach the problem. Participants 2 and 16 suggest an approach that students may take:

> "The thinking about the result is probably in two tiers: (1) which entities do I need, and (2) which attributes of these entities do I need. So if the question asks for the IDs of customers in the result, which it more or less does, then the reflex is to ensure that the record for customer is in the result, and from that select that the ID."
> - Participant 2

> "The question is for the *customer* IDs. Therefore, I think the student automatically decides to use the customer table (and the transaction table for the other part of the question), because that is the table that has information on customers. Not realizing that ID's are often stored in other tables as well. Possibly this also points to a very rigorous application of a step-by-step solution plan, without thinking what you are doing." - Participant 16

The first two hypotheses reflect a template-based way of working. The template (or step-by-step work plan) comes out of long-term memory when it seems applicable to the question, but the reuse of queries draws from similarity to a recently answered question, which is closer to short-term memory. Therefore, one of these hypotheses is classified under *generalizations*, whereas the other falls under *procedural fixedness*, a category that is closely related because it focuses on the student becoming accustomed to writing one type of query. Other hypotheses capture *lack of practice* (2), *mental model* and *ignorance*. Finally, one hypothesis is classified as *not an error*.

The hypothesis of the students is similar to the two most agreed-upon hypotheses by experts. They both suppose there is some query template being applied here, either from memory or from previous questions.

5.5.4 Missing second table for self-join

Provided problem statement

Question:
List all pairs of customer IDs who live on a street with the same name but in a different city.
Error context:
Here, the student should have applied a self-join, but only took the customer table once.

Student answer:

```
SELECT cName, cName
    AS name_1, name_2
FROM customer
WHERE (street_1 == street_2
AND city_1 != city_2);
```

Findings for this error are displayed in Figure 5.5. For this error, we see a high level of agreement between experts. For the first three hypotheses, none of the experts disagreed. What makes it even more interesting is that the answers represent a wide variety of hypotheses on misconceptions. We see misconceptions about tables/relations, restrictions in the FROM clause, the WHERE clause, and attributes. Participants also mention possible influences of programming:

> "In the world of programming languages, such a construct would be implemented with two nested iterators defined over the same (shared) data structure. Students may apply this concept to tables as well." - Participant 18

Some of the hypotheses in Figure 5.5 explore a more black-box mental model of the DBMS, such as the top hypothesis. They reflect incomplete knowledge on the part of the student regarding how tables are accessed. Incomplete knowledge matches the student misconception by Miedema et al. [111]. Their other proposed misconception for this error is misunderstanding the scope of elements in a query, which is mostly agreed upon by the experts.

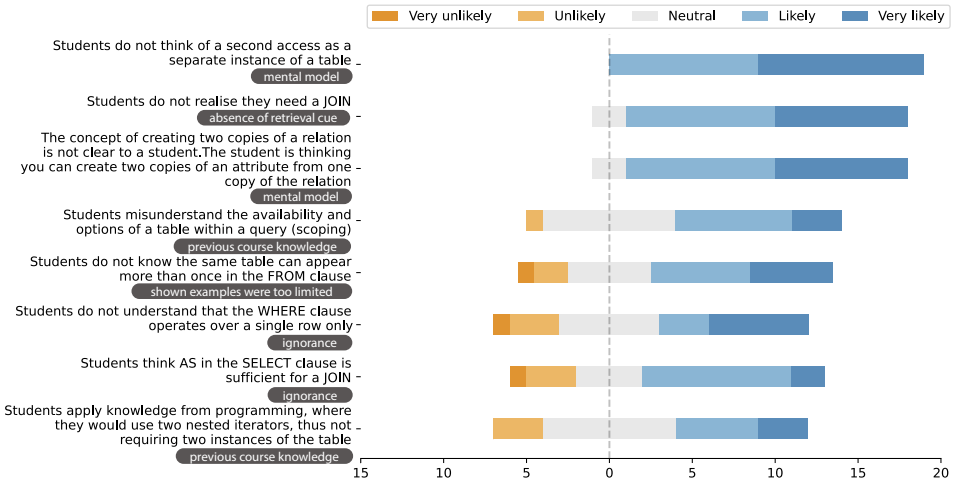


Figure 5.5: Expert hypotheses for *Missing second table for self-join*

We categorize two of the expert hypotheses as *previous course knowledge*, two as *mental model* issues, two as *ignorance* and one as *shown examples were too limited*. Finally, as Taipalus maps this error to *absence of retrieval cue*, we apply that category for the hypothesis *Students do not realize they need a JOIN*.

In the extra comment space on this error, Participant 1 mentioned:

> "I would also think that because SQL allows for 'implicit' joins via the WHERE clause, the students do not clearly see whether something is a filter on a row, vs. a join condition. I'd blame this on SQL's syntax, not on the students."

Such an error is also partially due to the teaching style of the student's lecturers and instructors. Some are strict on teaching explicit JOINs whereas others mostly teach implicit JOINs through Cartesian products.

5.5.5 Lack of understanding of when to apply GROUP BY

Provided problem statement

Question:
Return the stores table ordered alphabetically on city.
Error context:
The student added GROUP BY when it was not required, leading to an incorrect answer.

Student answer:

```
SELECT city, street, sName, sID
FROM store
GROUP BY city
ORDER BY city ASC;
```

Findings for this error are displayed in Figure 5.6. All hypotheses for this error skew towards neutrality, with many experts both agreeing and disagreeing with each hypothesis. Perhaps the error title was too general, and we should have focused on this specific error context. Nevertheless, we received some interesting insights from the first questionnaire:

> "Student expecting relationship between GROUP BY and ORDER BY, possibly caused by the misinterpretation of examples where both co-occur (I guess it's not unlikely to have ORDER BY clauses over columns that have been grouped, because groups are often the 'primary' sort key – which of course doesn't imply that GROUP BY is a pre-requisite for ORDER BY)." - Participant 9

Likely, a biased selection of examples sampled by the instructor (or read by the student) introduces misconceptions, as such examples have a significant impact on the students' mental model of the concept.

> "[...] the student could have seen something was wrong with their answer by seeing that the select statement asks values from non-grouped columns, leading to a malformed query. So maybe the biggest misconception is that you can group on a column A and then still project values from a different column B (depending on the DBMS this is only true if B has the same values for every identical value of A)." - Participant 16

Participant 16 focuses on how the student could have evaluated their result and therefore fixed their problem. As the student apparently did not do that, they must have a misconception of the order of operations during SQL execution. This leads us to the previously identified misconception of misunderstanding SQL syntax and its internal (in)consistency in [111]. The student may not have realized that after applying GROUP BY, not all actions are possible anymore, which is also reflected in the top hypothesis by the experts. Furthermore, the explanation of Participant 16 shows that the student did not reflect on the result table. This may show the ignorance that Taipalus maps this error to [172]. Although different wording is used in all three cases, the students and experts most likely feel similarly about this error.

Mapping of the hypotheses reveals that various themes were touched upon: we categorized two hypotheses as *absence of retrieval cue*, two as *shown examples were too limited*, one as *ignorance* (as in Taipalus' work mentioned above), one as *language* and one as *mental model*.

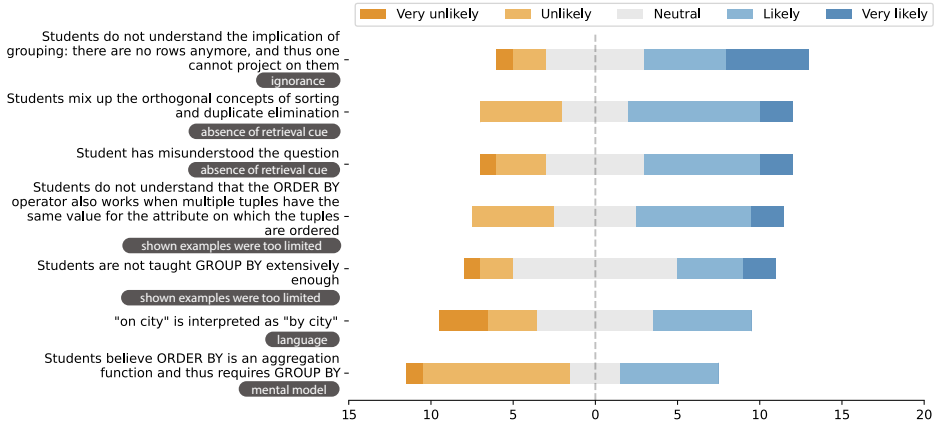


Figure 5.6: Expert hypotheses for *Lack of understanding of when to apply GROUP BY*

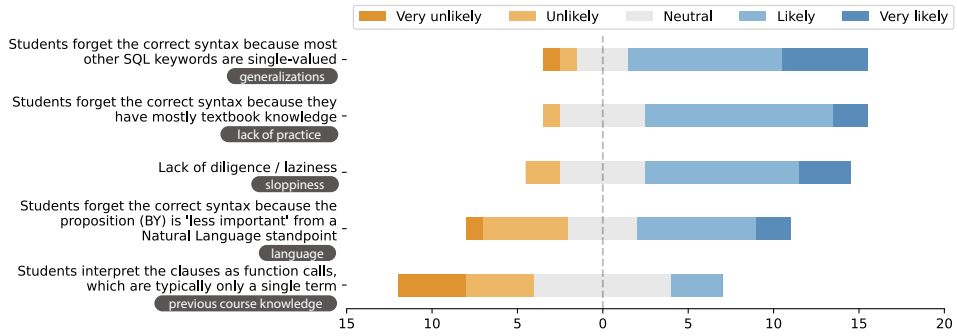


Figure 5.7: Expert hypotheses for *Missing parts of keywords*

5.5.6 **Missing parts of keywords**

Provided problem statement

Question:
Return the stores table ordered alphabetically on city.
Error context:
Missing parts of keywords (GROUP BY, ORDER BY, INSERT INTO). The student wrote ORDER instead of ORDER BY.

Student answer:

```
SELECT *
FROM store AS s
ORDER s.city ASC
```

Findings for this error are displayed in Figure 5.7. In round one, most of our participants mentioned students being careless, not knowing the syntax, or not remembering it correctly. For example, Participant 3 writes:

> "Student simply forgot the concrete syntax of SQL (where ORDER BY is a single token)." - Participant 3

The first four hypotheses in Figure 5.7 reflect this sentiment, although for different underlying reasons. Hypothesis 5 is the only one that falls outside of this category, but the large number of votes for (highly) unlikely shows that this is an unpopular opinion. Regardless of the hypotheses having similar sentiments, they do represent different types of underlying causes: we mapped them to *previous course knowledge, generalizations, language, lack of practice* and *sloppiness*.

This is the first error that the students and experts interpret in different ways. For the experts, the focus is on commonalities (Hypothesis 1) and practice (Hypothesis 2). However, from the students' perspective, this may have more to do with Natural Language effects. Although this hypothesis was posed by the experts, many of them disagreed with this statement.

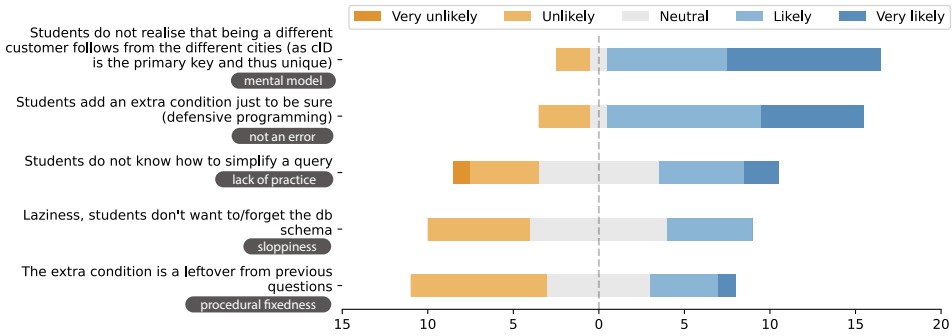


Figure 5.8: Expert hypotheses for *Extra condition when not required*

5.5.7 **Extra condition when not required**

Provided problem statement

Question:
List all pairs of customer IDs who live on a street with the same name but in a different city.

Error context:
In this case the participant added cID <> cID even though that could not occur with our schema.

Student answer:

```
SELECT c1.cID, c2.cID
FROM customer c1, customer c2
WHERE c1.cID <> c2.cID
AND c1.street = c2.street
AND c1.city <> c2.city;
```

Findings for this error are displayed in Figure 5.8. The most agreed-upon hypothesis reflects on either students' conception of the schema in general, or their conception of foreign keys. The other four hypotheses focus on the process of writing and editing. Participant 2 explicitly focuses on the writing process and working memory in their round one answer:

> "Mental compartmentalization: the student wants to ensure that (1) the two customers are not the same customer and (2) have different city fields. And they do not realise that (1) follows from (2). What in my experience also plays a role is that in another question there was also asked for two <something> where it was important they were distinct, and where this did not follow from the additional selection criterium." - Participant 2

The hypotheses of the experts again capture a broad range of categories: *mental model, procedural fixedness, lack of practice, sloppiness* and *not an error* all occurred once.

An extra remark from Participant 7 leads us to believe that the hypothesis on laziness and remembering the schema could have had more experts agreeing:

> "I actually kind of like parts of [...] "students forget the db schema", but I cannot get behind the value judgement of "Laziness"" - Participant 7

This is a valuable comment. In retrospect, we should have framed this hypothesis in a more discerning manner.

Regarding Q2, Miedema et al. [111] identified this as lacking knowledge of primary keys, a hypothesis that has not been explicitly mentioned by our experts. However, primary keys are mentioned in the highest-ranking hypothesis. Therefore, although the angle is not the same, both experts and students attribute this error to primary keys.

5.5.8 Ambiguous column name because of missing alias

Provided problem statement

Question:
Find the names of all inventory items that have a higher unit price than Bananas.
Error context:
It is not clear which pID is meant in pID=pID, because the participant did not add the alias.

Student (partial) answer:

```
SELECT unit_price
FROM inventory
INNER JOIN product
ON pID = pID
WHERE pName = "Bananas"
```

Findings for this error are displayed in Figure 5.9. In the first three hypotheses, the experts propose that students assume that the DBMS works similarly to a human brain. In that case, if something is interpretable to the user, it should also be clear to the DBMS. This explains why the student missed that there is ambiguity in the query. As Participant 8 says it:

> "Not double checking all of the details – assuming that the query has the same context as the query writer's brain." - Participant 8

For this error we have only four hypothesis, over three categories: *mental model* (2), *ignorance* and *lack of practice*. In the case of the hypotheses here, these three categories are closely related: a lack of practice may lead to an incorrect mental model, and ignorance may lead to similar mistakes.

The student may not have been able to evaluate the query from the correct perspective because of a lack of knowledge or experience. This is the explanation given by Miedema et al. [111] for this error. It is not a misconception in itself, but more of an explanation why the misconception hypotheses as suggested by the experts occur.

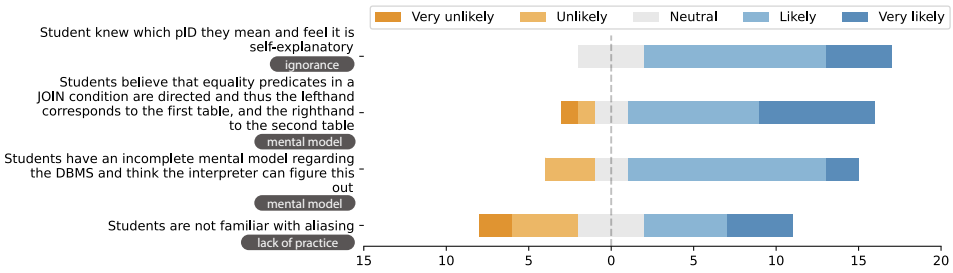


Figure 5.9: Expert hypotheses for *Ambiguous column name because of missing alias*

5.5.9 Using comma instead of AND in WHERE clause

Provided problem statement

Question:
Find the names of all inventory items that have a higher unit price than Bananas.
Error context:
Error occurs in the main query (comma after i.unit_price) but not in the subquery.

Student answer:

```
SELECT p.pName
FROM inventory as I, product as p
WHERE (SELECT i.unit_price
FROM inventory as I, product as p
WHERE i.pID = p.pID
AND p.pName = "Bananas")
    < i.unit_price,
p.pID = i.pID;
```

Findings for this error are displayed in Figure 5.10. In Round 1, many of our participants mentioned that this mistake comes from sloppiness or confusion of what the student was writing where, signaled by to lack of indentation. For example, Participant 13 writes:

> "Possibly a careless mistake. The student knows how to write a where clause but probably got confused due to the way he/she placed the subquery in the where clause before the join over pid. That is, the mistake possibly arises from poorly-structured where clause in the main query." - Participant 13

In Round 2, the experts benefited from the introduction of other experts' opinions and found other hypotheses relatively likely too. These describe how existing knowledge interferes with learning and the writing process. We can categorize the hypotheses as follows: two hypotheses reference *previous course knowledge*, two hypotheses reference *generalizations*, one is related to *language*, and one relates to *sloppiness*.

Regarding Q2, here is another agreed-upon hypothesis that matches the findings of Miedema et al. [111]. They attribute this error to a misunderstanding of SQL syntax and its internal (in)consistency, where the inconsistency is between the SELECT, FROM, and WHERE clauses. This hypothesis is similar to the 'list clause' hypothesis, which is broadly agreed upon by the panel of experts.

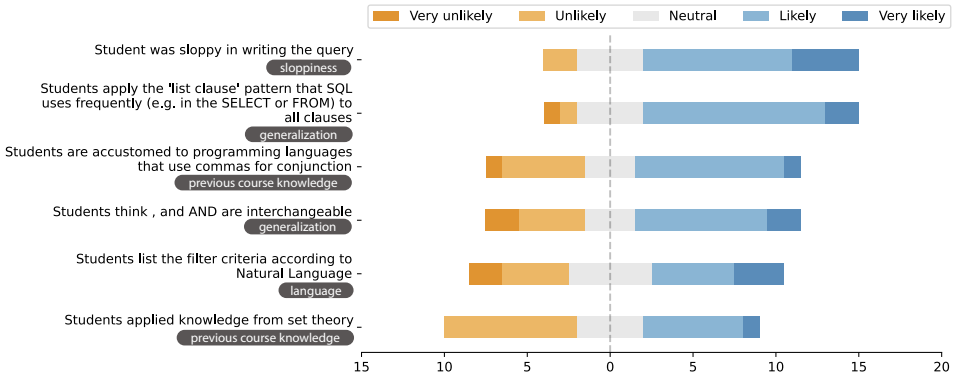


Figure 5.10: Expert hypotheses for *Using comma instead of AND in WHERE clause*

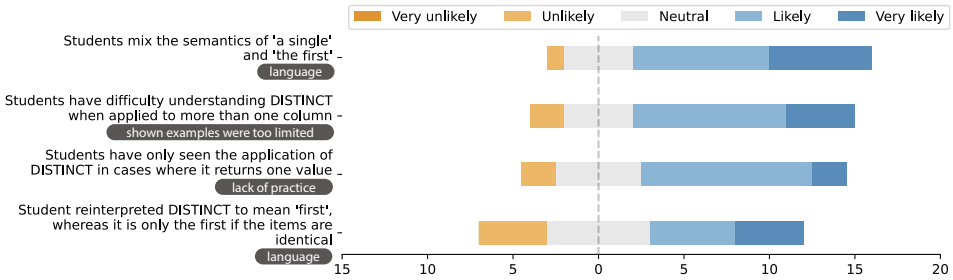


Figure 5.11: Expert hypotheses for *Thinking DISTINCT will take the first item of a list*

5.5.10 **Thinking DISTINCT will take the first item of a list**

Provided problem statement

Question:
For this question: Find the names of all inventory items that have a higher unit price than Bananas. There was a follow-up question: What happens if there are two products named bananas?

Student's original query:

```
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)
```

On the question *What happens if there are two products named Bananas?* they then elaborated:

> "[Suppose] There will be two prices [for two bananas]. 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. And then it would just compare [the price] with that one."

Findings for this error are displayed in Figure 5.11. In Round 1, most of the participants considered misunderstandings, carelessness, or low familiarity as reasons for this error. It is in such cases that the Delphi methodology is advantageous: there were some participants who went beyond conclusions of carelessness, and the inclusion of their hypotheses led our other participants to reflect on a deeper level.

For example, Participant 4 questioned the examples that were used to teach DISTINCT:

> "Was the example with which they were taught distinct done using just one type of distinct value?" - Participant 4

This questioning led to the inclusion of Hypothesis 3, as did the following text from Participant 2:

> "A DISTINCT does have the connotation of reducing a multitude to something smaller, so if I get many, and only want it to be one (so fewer), it sort of seems to come close to that. The student may have seen an example where a query returned a result with all duplicates, and the DISTINCT was used to reduce that to one record, which comes close to what they want here." - Participant 2

Overall, these four hypotheses capture three categories of causes: *language* (2), *lack of practice* and *shown examples were too limited*.

We have seen this focus on examples already, suggested by our experts for the errors with GROUP BY. It was also suggested by Miedema et al. [111], who described the misconception as *Believing DISTINCT takes the first item of a list*. As the upper three hypotheses in the figure are ranked similarly, we can conclude that the students and experts agree on this hypothesis.

5.5.11 Lack of understanding of subqueries

Provided problem statement

Question:
List all customer IDs, dates and quantities of transactions containing products named Apples.
Error context:
There is no need for the authors of these queries to write a subquery. It increases complexity and reduces readability.

Student answers:

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

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

Findings for this error are displayed in Figure 5.12. For this section, our participants debated whether this was actually an error. Do we expect our students to consider the efficiency of their queries? Participant 18 did not think so:

> "Not sure why this is considered an error. Semantically, the query is correct. Subqueries might execute slower in some DB engines, but I would not expect the students to know this." - Participant 18

Participant 6 agreed, arguing that efficiency depends on which DBMS is used:

> "I do not agree with the premise of the question. The efficiency argument is not robust: this varies on different DBMSs. In fact there are DBMSs where the subquery

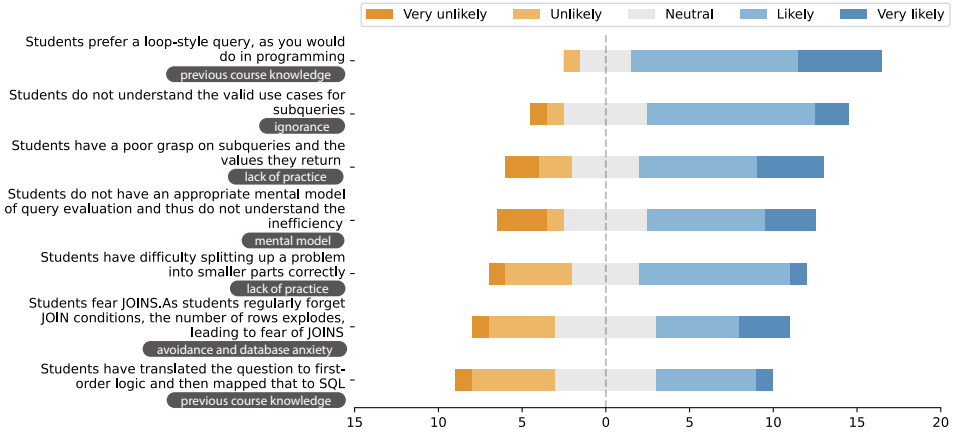


Figure 5.12: Expert hypotheses for *Lack of understanding of subqueries*

> variant is actually more efficient, and so it might be that the student was exposed earlier to information from that context." - Participant 6

However, these participants only focused on the efficiency of this example and did not touch upon the readability of subqueries. Two experts who did, are participants 1 and 16.

> "Subqueries using identifiers from the upper level query are hard and confusing. They are simply difficult to read." - Participant 1

> "Honestly, I find (especially) the second query immensely readable. I prefer subqueries over joins mostly for readability. This is a common split among students as well. About a third typically like subqueries, while the majority prefer joins. If you consider the (previously introduced) idea that there can be multiple product with the same name, then I really like the second query." - Participant 16

These participants disagree on whether the queries presented here are readable or not, which might have to do with the query's size.

Participants also mentioned translating queries, either from Natural Language (Hypothesis 5) or First Order Logic (Hypothesis 7). The plot in Figure 5.12 does not indicate high agreement within the participant group, but it represents an interesting question nonetheless: How do students decompose a more complex problem, and where do they start the formulation of their query?

Overall, these seven hypotheses can be grouped into five categories: *previous course knowledge* (2), *lack of practice* (2), *ignorance, mental model* and the new category of *avoidance and database anxiety*.

Regarding Q2, Miedema et al. [111] did not attribute this error to a specific misconception, but instead explained this by a lack of knowledge or experience. Given the discussion by experts of whether this should even count as an error, and whether that is DBMS-dependent or not, a lack of student insight could be a fitting explanation.

5.6 Discussion

The method that we applied in this study resulted in capturing more input per error than presented by Miedema et al. [111]. Each error had between four and eight misconception hypotheses. In the mapping of these hypotheses to causes introduced by Miedema et al. and Taipalus [172], we found that they covered several different topics. Each error had hypotheses from three to six categories, scaling with the number of hypotheses. We found representations of all classes identified in previous work, except for *working memory overload*. This is likely due to the setup of the experiment, providing small snippets of information to our experts without the broader context. Beyond these classes from literature, there were representations of the classes *lack of practice*, *sloppiness* and *not an error*.

In this chapter, we also introduce two new causes of errors. The first is *shown examples were too limited*. This reflects a lack of depth in the material that the teachers use to introduce concepts to their students. Examples may be too simplistic, and as such do not showcase the full range of opportunities for the concept. The second is *avoidance and database anxiety*. This is the database equivalent of *math anxiety*. Math anxiety has been shown to disrupt cognitive processes by monopolizing resources of working memory [9]. A similar process may occur in SQL querying when students need to apply more complex concepts such as JOINs, aggregation, and subqueries. Future work should examine the extent to which database anxiety influences student performance.

5.6.1 Implications for Teaching

To improve upon our teaching of SQL, the first step is to identify where possible problems in knowledge transfer occur. This identification of faulty transfer, through the uncovering of misconceptions, is an indication of where course material and instruction need to be adapted to the students' previous knowledge. Teachers who have an understanding of misconceptions can work towards knowledge refinement and reorganization, to build students' knowledge of the taught concepts [163].

One central theme that came up regularly in our findings was both a lack of practice and a lack of appropriate examples demonstrating SQL concepts. Such issues can be relieved by increasing the opportunities that students have to interact with SQL, as Participant 4 expressed:

> "Students need more practical experience, just like any other programming language. We need to stop treating databases as an extension of math."

Typically, in Computer Science curricula, there are several courses centered around programming. Students are taught about programming languages, design patterns, algorithms, and data structures, and they get the opportunity to practice their programming skills through several courses and projects. Why then do we teach only the basics with regard to Databases? With the increasing need for Data Science topics in Computer Science curricula, there should be more focus on teaching students about data storage and retrieval.

5.6.2 **Comparing Prior Theories and Research Findings**

Existing research is conflicted about whether teachers and other experts have enough insight to discuss their students' misconceptions. Goldman et al. find that teachers have an incomplete understanding of student learning [63]. From a practical perspective, Qian et al. discuss the latent nature of misconceptions, and how this may make it difficult for teachers to detect them [143]. Does this then mean that educators have no insight into where students may misinterpret material? Brown et al. [27] instead hypothesize that although educators are not accurate in predicting the frequency of (programming) mistakes, they may still be accurate about the cause of mistakes and a student's conception of the mistake [27]. To validate this, we recommend finding the prevalence of the misconceptions presented in this chapter in a future study.

5.6.3 **Strengths and Limitations**

Our main limitation is the number of errors we consider in this chapter. As participants in any study should not be overwhelmed, and our format required open-ended questions, we decided to limit ourselves to 11 errors. Although these errors have been shown in section 5.3 to be common and persistent, there are many additional errors that we could have examined, generating more hypotheses for misconceptions. Future work should consider examining misconceptions for these errors. Additional hypotheses might have also come up if we had more participants in our study.

Although our study only included two rounds versus the typical three or more of traditional Delphi, we do not think adding extra rounds would have changed our findings. An extra round could show how our participant population thought of the various hypotheses, such that participants could change their minds. But, as our focus is on qualitative over quantitative results, the scores themselves are not the most important thing, the hypotheses and quotes are.

Instead of the Policy Delphi approach we adopted, we could have also run a focus group. An advantage of Policy Delphi over a focus group is that it allowed asynchronous 'discussion' between participants on various continents, with different time zones, who may not have been able to match up schedules otherwise. Furthermore, the anonymous nature of Delphi was an advantage to make sure that each voice counted equally.

The anonymous approach of our participants with regard to filling in the survey means that we did not have the exact same group of respondents for Rounds 1 and 2. We cannot tell to which extent our groups overlapped, although the different number of respondents indicates that the groups do not match. Although Delphi is about anonymity, the participants only need to be anonymous to the other respondents, not necessarily to the researchers. However, we do not think this is a threat to validity, as our main contribution is the list of hypotheses presented above. This is a result of Round 1 only, and thus the differing groups do not have a large impact.

Finally, we could have strengthened our results for Q1 by asking the participants to indicate which group they identified most heavily with (educator, researcher, practitioner), instead

of indicating all they identified with. As a result of this choice, we could not run statistical tests on our data to see if different groups had different opinions.

5.6.4 **Validity**

In this discussion, we follow the descriptions of validity for qualitative work by Joseph Maxwell [106]. He defines the nature of five dimensions of validity: descriptive validity, interpretive validity, theoretical validity, (internal and external) generalizability, and evaluative validity [106].

First of all, the setup of the study, with participants writing down their answers in a Google Form, means we record the participants' opinions as quotes. As such, our results are an accurate representation of our participants' opinions, meaning our study has high descriptive validity. Using quotes also means that we do not have to interpret what our participants mean with their actions, meaning the interpretative validity of the study is high too. One area where some interpretive validity may have been lost is the aggregation step between rounds one and two. An alternative setup to avoid this would have transferred all participants' quotes directly from the round one form to round two. However, this would have led to many overlapping answers, which would distort the hypotheses' voting results. As all three authors translated and aggregated the answers in an iterative process, and the participants voted on the newly generated hypotheses, we believe the interpretative validity of the study was upheld.

Given the direct reporting of participant perspectives in this chapter, presented as a stepping stone towards more research on SQL misconceptions, the authors believe that the concepts of theoretical and evaluative validity may not be applicable.

Finally, the generalizability of the study has been taken into account by inviting participants of varying backgrounds. The participants are of different geographical and cultural backgrounds, have different types of jobs with various levels of experience, work at different companies and institutions, and have differing ages and gender identities. This provides us with confidence regarding the external generalizability (to different communities). The study was not designed to have internal generalizability (the hypotheses do not apply to different errors), future work should consider possible misconceptions for errors that we did not discuss here. One note on the external generalizability is that student misconceptions most likely depend on the teaching method (types of examples, database schema, teacher preference, etc.). Therefore, not all misconceptions identified in a study such as ours will apply to all students. However, the significant hindrance that misconceptions cause, makes it important to keep all possibilities in mind.

6

Exploring Misconception Prevalences

With the misconceptions identified in chapter 4 and chapter 5, we are able to design more complete educational approaches. However, one big question remains: How representative are the misconceptions we identified? The limited geographical and cultural spreading among our participants might have biased our previous findings. As such, we set out to explore whether our findings also hold for a more diverse student population. We developed the Multiple-choice SQL Misconceptions Instrument (MSMI1), with pairs of questions aiming to elicit the same misconception. Then, we developed feedback on student answers, such that they could use the MSMI1 as a formative assessment. Finally, we deployed the instrument to students worldwide. We found that all misconceptions are held to some extent, with prevalence scores ranging from one to fifty-two percent of the participant population. Additionally, we have uncovered previously unidentified areas of struggle, allowing us to identify new misconceptions.

The contents of this chapter have previously appeared in **Miedema, D., Liut, M., Fletcher, G., and Aivaloglou, E.** "There is no ambiguity on what to return": Investigating the Prevalence of SQL Misconceptions. In *23rd Koli Calling International Conference on Computing Education Research* (Koli, Finland, 2023)[116]. The questions have been published as **Miedema, D., Liut, M., Fletcher, G., and Aivaloglou, E.** MSMI1: Towards a Validated SQL Misconceptions Instrument. In *Proceedings of the 2023 ACM Conference on International Computing Education Research V.2* (Chicago, USA, 2023)[115].

6.1 Introduction

Database Education as a research field has been on the rise [5]. Various studies have investigated the teaching and learning of the Structured Query Language (SQL); the errors students make have been identified and categorized [3, 139, 178], tools have been developed for supporting students in query writing [70, 112, 126, 127] and repair [104, 124], and visualizations have been proposed for assisting in query understanding [59, 100, 112]. Several existing works have concluded that SQL, simple as it may appear, is challenging for novices.

Recent research has identified misconceptions as underlying causes of student challenges and errors. This line of work was initiated by Taipalus [172], who also discussed the potential causes behind persistent SQL errors [178], and was continued by Miedema et al. [111], who collected and analyzed qualitative data on the thought process of 21 students while they were working on query formulation problems. In their work, they identified fourteen misconceptions in four categories: misconceptions based on previous course knowledge, generalization-based misconceptions, misconceptions based on language, and misconceptions due to an incomplete or incorrect mental model [111].

This chapter builds on that work. It is inspired by the importance of research on understanding the problems novices face with SQL, and the design of interventions to support them. Currently, although a set of misconceptions has been identified for SQL, it is unclear how representative these misconceptions are for a wider population of students. This chapter aims to investigate the prevalence of a subset of previously identified misconceptions among a diverse student population, as well as the reasoning behind erroneous student answers. Our goal is to answer the following research questions (RQs):

RQ1: What is the prevalence of previously identified SQL misconceptions in a diverse student population?

RQ2: What are students' reasonings when exhibiting misconceptions?

To answer our RQs, we created a set of multiple-choice questions (MCQs), similar to previous studies on misconceptions in other computer science domains [134, 171, 204]. Through our questionnaire, we tested 12 of the misconceptions identified by Miedema et al. [111]. We included several documented safety mechanisms to ensure that participants' responses are based on their knowledge and are not guesses or accidentally correct answers (e.g. certainty of response and an alternative question for each misconception). Question dependence was subsequently validated to ensure that each MCQ pair was capturing the same misconception. Moreover, we included an open-text field per question, for participants to elaborate on their answer selection, which provided us with additional qualitative data. The questionnaire was administered to 249 participants from six universities in four different countries on three different continents.

Our results show that all identified misconceptions were held to some extent, with prevalence scores ranging from one to 52% percent of the student population. Moreover, we discover previously unidentified areas of struggle, leading to the identification of ten new misconceptions.

6.2 Related work

6.2.1 Testing for Misconceptions

Multiple-choice questions are a common way to test students' knowledge, both within a classroom environment and a research setting. There are many advantages to MCQs, such as ease of (volume) grading, heightened student confidence, being able to cover a wider range of topics than a traditional test, and (seeming) objectivity [97]. However, although students' performance on MCQs and constructed-response questions correlates, exclusively using MCQs to evaluate student performance in a classroom setting may not give the right perspective on a student's skills [97]. For example, students are not able to show their thought process, reason outside of the box, and cannot gain partial credit.

To reduce this effect, we can improve the design of MCQ tests, while maintaining their advantages. Tamir introduced the concept of two-tier MCQ tests, which add a justification question on top of each original question [181]. This justification can help to assess learning and misconceptions, and students can use it to illustrate their thought processes. Alternatively, we could add a *certainty of response index* [71], which gives us information on whether our students are guessing the answers to the test, or actually (think they) know the answers. We can also combine these approaches into a three-tiered design as defined by [191]. Finally, to reduce the effect of guessing, we can ask students to select all answers that they think are correct. If we also have multiple correct answers, getting all correct answers right and not selecting any erroneous answers reflects mastery of a topic.

Various MCQ test designs have been used to test programming knowledge. For example, Whalley et al. used one-tier MCQs to test programming comprehension [195]. Ma examined students' understanding of program execution and assignment statements using MCQs and students' notes [102]. Swidan et al. used MCQs to identify misconceptions and included the *certainty of response index* [171].

Another application of MCQs in testing are Concept Inventories (CI): highly researched tests taken by students to investigate whether they understand the central concepts of an area of study. The first CI was the Force Concept Inventory [78], which centered on Newtonian Laws. More recently, CIs have been under development for various subfields of Computer Science. For basic programming knowledge, the FCS1 [182] and SCS1 [133] were developed, Herman developed a CI for Digital Logic [73], Porter et al. created one for Basic Data Structures [137], and Wittie et al. worked on a CI for CS2 [196]. Research outcomes of applied MCQ studies can inform the development of future CIs.

6.3 Method

Our aim in this chapter is to identify the prevalence of misconceptions on SQL in a widely varying student population. To this end, we created a multiple-choice questionnaire with questions that should evoke the misconceptions identified by Miedema et al. [111]. We used MCQs as they have been shown to work for identifying misconceptions for programming languages [101, 160, 171]. They also make it possible to run the questionnaire on a large scale. Additionally, the questions should exclusively evoke the misconceptions we are exploring. Open questions could make it difficult to quantify this.

Some of our questions had more than one correct answer. These are only counted as correct if the participant has exclusively selected correct answers. If not all correct answers are selected, we mark answers as incomplete.

6.3.1 **Participants**

The authors of this chapter reached out to database course instructors within their network, to identify courses in which the questionnaire could be applied and helpful for the students. Instructors then shared the questionnaire with their students as study material for the exam. To add value for the participants, the questionnaire was set up as preparatory material. This meant that, upon completion of the questionnaire, the participants received an overview of their answers, the correctness per question, and some feedback on incorrect answers.

This study had 249 participants from six universities in four different countries on three continents. We have gender data for 230 of these participants, where 195 participants identified as male, 45 identified as female, four participants preferred not to say, four identified as non-binary and one participant self-described their identity. The median age of our participants was 21 ($\mu = 21.43, \sigma = 2.79$), with a minimum of 18 and a maximum of 45.

Ethics. The participants were not financially compensated for participation in the study. The study design and data collection were approved by the Ethical Review Board of Eindhoven University of Technology.

6.3.2 **Materials**

For the questionnaire, we selected 12 different misconceptions from Miedema et al. [111]. We introduce four question types:

1. Fill in the blank in the query, where options to fill in the blank were given. The participants could select more than one answer.
2. Select the correct query for the given natural language question. The participants could select more than one answer.
3. Given a natural language question and a SQL query, the participants were asked to evaluate whether this was the correct translation.
4. Given a natural language question and a SQL query,the participants were asked a meta-question about the query.

As we wanted to make sure we were measuring misconceptions, instead of guesses or accidental correct answers, we introduced three safety mechanisms. First of all, for each of the misconceptions tested, we generated two different questions that should evoke the misconception. These were typically of two different question types as mentioned above. The questions were not asked pair-wise but mixed up with questions on different misconceptions as long as these questions would not give hints on the correct answer to the following questions. To validate that the questions were capturing the same misconception, we calculated the dependence between the pairs with a χ^2 test.

Second, for each question, we asked how certain the participant was of their answer on a Likert scale from one through seven. This allowed us to disregard those answers that indicated a misconception but were shown to be guesses. After the exclusion of guesses, we calculated the aforementioned dependence between each pair of misconceptions.

Third, we offered the participants a text field in which they could elaborate on their answers. They were asked the question "*Can you elaborate on why you think it is correct?*". These answers were used to strengthen our interpretations for the particular misconception we were evaluating, as they regularly reflected faulty logic in the participants' thought processes.

Finally, all questions were checked by four SQL educators who were not on the research team, to test for clarity of formulation, non-determinism of answers, plausible distractors, and possible coding errors.

Ten of the pairs of questions used in this questionnaire have been published previously as the Multiple-choice SQL Misconceptions Instrument (MSMI1)[115], with the exception of two pairs of questions that were found to be independent.

6.3.3 **Pilot Study**

To decide how many SQL misconceptions could reasonably be addressed during the study, and to make sure no problems were introduced upon digitizing the questionnaire, we ran a small pilot study.

For this pilot, we used the materials described above. However, instead of asynchronous administering of the questionnaire, we recruited a group of nine participants from the same target group as the main study and colocated them to simultaneously take the questionnaire.

The pilot started with the researcher explaining the purpose of the questionnaire: testing the participants' SQL knowledge. The participants were asked to take the test, focusing on both speed and accuracy. Then, if they found something they did not understand, or identified possible mistakes, they were asked to raise their hand to share it with the experimenter. During the pilot study, we found that the questionnaire as written took our participants around 60 minutes and contained no major problems. We corrected minor errors, finalized the questionnaire, and continued with recruitment and running the main study.

6.3.4 **Data Analysis**

For the quantitative data, we encoded each answer as either correct, erroneous, incomplete, or containing a misconception. Various questions had the option to select more than one answer. For these questions, we coded the participant as 'having' the misconception if any of their selected answers contained the misconception. From this set we removed all participants that indicated they were unsure about their answer, counting only the ones that indicated a certainty of five or higher on a Likert-scale from one to seven.

We analyzed whether the pairs of questions could be considered dependent: whether a misconception in question one also meant a misconception in question two. We also checked if the misconceptions were dependent on one another; whether holding misconception one meant also holding misconception two. This was done through χ^2 tests.

	BRACKETS	NEQ	IS	SCOPING_WITH	DISTINCT	MISSING_ALIAS	PK_DISTINCT	UNDERSTANDING_PK	EQ_PK	ALIAS_SYNTAX	SCOPING_SELFJOIN	MISSING_JOIN
BRACKETS	x											
NEQ		x										
IS	**		x									
SCOPING_WITH				x								
DISTINCT					x							
MISSING_ALIAS			*	*		x						
PK_DISTINCT	**			*			x					
UNDERSTANDING_PK	*		*				**	x				
EQ_PK									x			
ALIAS_SYNTAX	**	*								x		
SCOPING_SELFJOIN	**					**	*	*			x	
MISSING_JOIN			*				**	*	*		**	x

Table 6.1: χ^2 scores between misconceptions. * for a p-value between 0.05 and 0.01, ** for a p-value <0.01.

The qualitative coding was done in pairs. The first author explored the free text answers to identify codes to start from. Then, the set of misconceptions was split in two, with each set independently being coded by two authors. Each author could create any new and extra codes that they felt were necessary. After coding, the pair of authors discussed any new codes they introduced and resolved any conflict between their coding.

6.4 Results

In this section, we discuss the qualitative and quantitative results for the misconception categories in our questionnaire. Each category has its own section below. For an overview of our findings, you can find the misconception prevalences in Table 6.2, the relations between the misconceptions in Table 6.1, and the coding summary of qualitative answers in Table 6.3. Please note that all quotes that are printed in this chapter are literal quotes and may contain spelling mistakes.

General results. Before we dive into each category, we make some general observations.

The χ^2 tests of independence revealed that ten out of our twelve pairs of questions are dependent. The independent pairs are SCOPING_SELFJOIN and MISSING_JOIN. For the former, most students struggled much more with the second question than the first, so the questions may not have been of equal difficulty. For MISSING_JOIN, we later found out that there were some problems with the question (see subsection 6.4.12).

On top of the dependencies within pairs, we also calculated the dependencies between pairs to see if students holding one misconception are more likely to also hold another. For all

Misconception	certainty	χ^2	Question 1 misconception	Question 1 error/ incomplete	Question 1 correct	Question 2 misconception	Question 2 error/ incomplete	Question 2 correct	%
BRACKETS	5.36	p=1.62e-13	21	0	228	32	0	217	11
NEQ	6.05	p=8.52e-70	17	112	120	13	102	134	6
IS	5.75	p=1.36e-99	71	3	175	70	2	177	28
SCOPING_WITH	4.60	p=1.62e-04	67	64	118	64	43	142	26
DISTINCT	4.82	p=8.83e-03	38	114	97	18	61	170	11
MISSING_ALIAS	5.46	p=4.31e-33	33	110	106	29	125	95	13
PK_DISTINCT	5.98	p=1.08e-06	84	NA	165	35	NA	214	24
UNDERSTANDING_PK	5.35	p=1.18e-06	10	17	222	15	122	112	5
EQ_PK	5.25	p=1.13e-03	38	29	182	76	18	155	23
ALIAS_SYNTAX	5.36	p=4.05e-11	1	56	192	3	123	123	1
SCOPING_SELFJOIN	5.16	p=0.105	24	34	191	46	68	135	14
MISSING_JOIN	4.37	p=0.232	144	NA	105	113	NA	136	52

Table 6.2: Counts per misconception question. pk stands for primary key, NA for not applicable. For the ten misconceptions in the top part of the table, the χ^2 score indicated that the questions were dependent. The bottom two are independent questions.

scores, see Table 6.1. We found nineteen pairs of misconceptions that seem to have a dependency, with eight of these being strong correlations.

Our codings for the qualitative answers are in Table 6.3. As can be seen in the table, of all elaborations on the misconceptions, only approximately half are informative, in that they mention something about the thought process of the student specific to the question. We analyze the question-specific answers per question in the sections below. The other half of the open answers can be divided in two. The majority of them are empty or filler answers ("nan", "nothing to say", "this is the correct answer"). Alternatively, they say something superficial about the thought process and decision-making, such as it looking familiar, being taught that way, or that it was a guess. Overall, the number of open answers that indicate guessing is relatively low.

6.4.1 Using parentheses on the WHERE clause (BRACKETS)

Definition: Students believe it is syntactically required to use parentheses around the contents of the WHERE clause.

As identified in Table 6.3, 11% of our participants believe that brackets are required in an SQL statement's WHERE clause. They have a high certainty score for this question, with an average score of 5.36 out of 7.

Three of our participants explained that the WHERE clause, according to them, requires brackets if it contains more than one condition:

> "there are multuple conditions" - Participant 116

We also found a reasoning that identifies a *new misconception*. Participant 84 writes:

> "For WHERE (...) since we use SQL query output as a table we need to specify the SQL query boundaries so that we know where it starts and ends. COUNT() is a function that requires certain parameters - these parameters are passed inside the parentheses"

So, they believe that a WHERE clause requires brackets when we use it to output a table. In this perspective, brackets seem to be used as a packaging behavior, marking the boundaries of what we do or do not use. In this sense, it may be related to the behavior required for a subquery.

Finally, participant 17 did not care much about where they could (not) remove parentheses:

> "Fairly uncertain. I will be using parentheses anyway for clarity of my future code, so I did not care too much about where can I omit them."

This is interesting, as not all DBMSs allow for extra brackets, and as such, this behavior may lead to syntax errors.

6.4.2 ≠ is valid syntax (NEQ)

Definition: Students believe that the ≠-sign is valid syntax to indicate inequality.

Students accepting the ≠-sign as valid syntax was relatively uncommon in this study with only 6% of participants making this mistake. However, this still seems to be a high count for participants who are using their computers, who have access to their keyboard, to see that ≠ is not easy to type. Additionally, the participants' certainty in this question was the highest of all, with an average of 6.05 out of 7.

For the 30 misconception-related answers received, we had 16 participants who gave a question-specific answer. Of these, 15 participants mentioned that they felt all answer options signaled inequality. For example, participant 36 wrote:

> "<>, != and the other sign of the selected options mean the same (i.e. "different than")"

As we can see, this participant did not stop to think that, although they knew the meaning of the symbols, they may not be valid syntax in SQL. Only one participant who selected the misconception answer did:

> "These all mean "unequal", I'm not sure if they can all be used in SQL though." - Participant 15

Vice versa, participant 275 prioritized ≠ over <>, thinking that the former is valid syntax:

> "the not equal symbol I think it is correct, the <> I know that means that its not equal but I am not 100% sure that we can use it in SQL"

6.4.3 IS and == are valid syntax to indicate equality (IS)

Definition: Students believe that the ==-sign and `is`-keyword are valid syntax to indicate equality.

Our prevalence scores show that the misconception answer was chosen 141 times over the two questions for 28% of the answers, with students being relatively certain about their answers (avg score of 5.75).

Many participants chose `is` as a correct answer, as they felt it was familiar. For example, participant 194 wrote:

> "I think IS can work like that, makes sense reading it aloud though, like 'city IS london' makes sense in english so maybe that translates to how it functions in SQL"

and participant 147 said that:

> "[...] IS just made sense because if we can't use it under this context then when are we going to use it."

Additionally, many of our participants felt that `is` and = are equivalent (36) or that `is` is a valid operator (2). For example, participant 51 wrote: "The operator "=" and "IS" is the exact same hence both will output the correct result."

This set of questions also covered the usage of == for equality, which is mentioned by Miedema et al. as a generalization from previous programming course knowledge [111]. Indeed, six

Code	Frequency									
	BRACKETS	NEQ	IS	SCOPING_WITH	DISTINCT	MISSING_ALIAS	PK_DISTINCT	UNDERSTANDING_PK	EQ_PK	SCOPING_SELFJOIN
Empty or filler answer	16	10	43	11	21	18	44	18	31	23
In the book/slides/lecture notes/ was taught this way	3		1				3			3
Looks familiar/right	2		7		7	1	1		2	
Guess/I don't remember/ I don't know	10	2	12	13	9	9	11	4	16	14
This was not taught/I have not used this keyword before			3	3	1					3
This is the only answer that makes sense/exclusion strategy				6	1	1		2	10	9
All answers are correct				5		2				
Repeats question			1	7		1	1		2	4
Misinterpreted question							6			
Question-specific reason	5	16	74	54	17	29	53	1	53	14

Table 6.3: Associated explanations for answers containing misconceptions were assigned one of the codes in this table. In section 6.4 we describe all answers in the category *Question-specific reason*.

participants referred explicitly to other programming languages, questioning whether this would generalize to SQL. For example, participant 263 wrote:

> "== is seen in many programming languages, but it might not be supported by SQL."

Interestingly, references to programming languages were also made for the `is` keyword:

> "I was never told about the possibility of using "IS", but it sounds like a reasonable choice (also - it works like this in Python <3)" - Participant 17

Furthermore, eleven participants wrote something about == being a valid operator. Finally, there were some participants who explained their answers with a statement suggesting that all three options (=, ==, `is`) indicate equivalence. Participant 33 wrote:

> "The second querie is wrong as it uses the different operator. All other queries are correct. The == is used in most programming languages as the conditional equality, but SQL also uses the = symbol, which at first I found strange. Using the keyword IS also works"

The question for this misconception was:
Which of the queries below answers the question: What are the IDs of customers who have ever purchased a product for the highest unit-price of all products?

The intended answer was:

```
WITH maxprice AS (
  SELECT MAX(unit-price)
    AS price
  FROM inventory)
SELECT t.cID
FROM transaction t,
    inventory i,
    transactionitem ti,
    maxprice
WHERE ti.pID = i.pID
AND t.tID = ti.tID
AND t.date = i.date
AND t.sID = i.sID
AND i.unit-price
    = maxprice.price
```

The misconception answer was:

```
WITH maxprice AS (
  SELECT MAX(unit-price)
    AS price
  FROM inventory)
SELECT t.cID
FROM transaction t,
    inventory i,
    transactionitem ti
WHERE ti.pID = i.pID
AND t.tID = ti.tID
AND t.date = i.date
AND t.sID = i.sID
AND i.unit-price
    = maxprice.price
```

Query 6.1: Question two for SCOPING_WITH, with the intended answer and the misconception answer.

6.4.4 Not including the Common Table Expression (CTE) name in the FROM clause (SCOPING_WITH)

Definition: Students believe that the results from the CTE are accessible without including the table in the main query.

Our prevalence scores show that this misconception was the third-most common, with 26% of the answers containing the misconception. It has a certainty score on the low end, with many participants close to guessing (the average score was 4.60 out of 7). One of the two questions in this pair is included in Query 6.1.

Out of the 131 misconception answers, we have 55 that present a question-specific reasoning. Approximately half of these are participants who were struggling with the question. Two participants explained that the question was too long for them (to read or understand), and 26 participants seemed to not have noticed the difference between the two versions of almost correct answers, one of which contained the misconception.

In the group of participants who did notice the difference (to (not) include the name of the CTE in the FROM clause), we identified three arguments:

1. Adding the name of the CTE does not make sense/does not seem necessary. This perspective was reflected by the explanations of four participants. Participant 24 wrote: “maxprice were defined in the beginning, so mentioning it in where clause did not make sense”

2. Adding the name of the CTE is redundant. Fourteen participants held this view. Participant 78 explained: "I believe you need to be comparing all ID values as the first two options do, but the second option includes maxprice in the FROM clause which is redundant"
3. Adding the name of the CTE is not allowed. Eight participants held this view. In their very short explanation, participant 248 wrote that: "'FROM maxprice' is not allowed"

Finally, this question uncovered a *new misconception* held by two participants (attending different universities). They argued that the CTE was not a relation, and as such could not be called in the FROM clause. This raises the question of what they think the CTE is, if not a relation, and warrants further research and interviews.

The question for this misconception was: Suppose we want to find the names of all customers who have a shopping list for the first day in January 2022 that such a shopping list exists. Is this a correct query to answer this question? You can assume that dates are encoded as strings in dd-mm-yyyy

The intended answer was:

```
SELECT cName
FROM customer c, shoppinglist s
WHERE c.cID = s.cID
AND date =
    (SELECT DISTINCT date
    FROM shoppinglist
    WHERE date LIKE "%-01-2022")
```

The answer options were:

1. Yes, the query is correct (misconception)
2. No, the query returns an incorrect answer (error)
3. No, the query has a syntax error (correct)
4. It is impossible to determine whether the query is correct (error)

Query 6.2: Question one for DISTINCT.

6.4.5 DISTINCT will take the first item from a list (DISTINCT)

Our prevalence scores show that this misconception was on the rarer end of the spectrum, with 11% of the answers. It has one of the lowest certainty scores too, although the participants are still well above guessing (with an average score of 4.82 out of 7). One of the questions of this pair is available in Query 6.2.

Miedema et al., hypothesized that this mistake originates from having only a few examples illustrating what DISTINCT does, leading to incorrect extrapolations of that knowledge [111]. This misconception was also described in a different form by Brass and Goldberg as *subquery term that might return more than one tuple* [26].

For this question, we have found relatively few question-specific elaborations. Many participants indicated they guessed or found the answer by excluding other answers. We did find two who literally stated that DISTINCT takes the first item from the list, as indicated by Miedema et al. Participant 118 understands that DISTINCT filters, but also believes it takes the first item:

> "Since order by defaults to ascending order I believe any of these answers are correct. MIN will produce the min[,] distinct will produce the first distinct lowest sID[,] and s.sID will produce the first lowest."

Eight participants did not notice that the subquery returns multiple answers, or did not realize that = does not accept a list of inputs. For example, participant 62 notices that the subquery uses LIKE, and this will return a list of outputs, but does not connect this to its impact on the main query:

> "The like operator is going to return all results with that same type."

Taking this further, participant 36 showed us a *new misconception* about subqueries. They believe that using the structure of `attribute = (subquery)` will allow the main query to take the first item from the subquery's result table. They write:

> "This query will return the names of customers who have made a shopping list for first of Jan, because % matches the all dates of Jan 2022, but date = takes only the first "

Another *new misconception* came from participant 84, who believes that "% sign allows us to use current date". This is not so strange in and of itself, because many programming environments allow us to use placeholders to retrieve 'variables' from the real world. However, this does not explain what they think of the LIKE keyword next to it.

The question for this misconception was: Find the IDs of customers from Berlin who have made at least 5 shopping lists for different dates. What is the correct expression to fill in the blank? (select one or more)

The intended answer was:

```
SELECT DISTINCT [ ]
FROM customer c, shoppinglist s
WHERE s.cID = c.cID
AND c.city = 'Berlin'
GROUP BY s.cID
HAVING COUNT(DISTINCT s.date) >= 5
```

The answer options were:

1. cID (misconception)
2. s.cID (correct))
3. c.cName
4. c.cID (correct)

Query 6.3: Question one for MISSING_ALIAS.

6.4.6 Missing alias in the SELECT clause (MISSING_ALIAS)

Definition: Students believe that it is not necessary to indicate which table they want to use an attribute from, and thus forego the required alias.

Our prevalence scores show that this misconception was not so common, with 13% of the answers, but, the certainty score indicates our participants were relatively sure of their answers (an average of 5.46 of 7). Question one for this misconception can be found in Query 6.3.

Eighteen of the explanations indicate participants' beliefs that all of the answer variants point to the correct attribute (cID, s.cID, c.cID for Q1, sName, s.sName, p.sName for Q2). However, the one without an alias prefix will not work, given that the cID/sName will be ambiguous. We also identified three *new misconceptions* through this question.

1. According to some of our participants, the JOIN condition *will remove the ambiguity* that comes from not using the alias. For example, participant 196 writes that: "there is no ambiguity on what to return because we test c.cid = s.cid". What this means is not entirely clear. Perhaps, to these participants, the JOIN condition creates a literal connection between the two tables/attributes as it would for a variable or a key-value store, such that calling the one also calls the other. This misconception was held by six participants and indicated seven times in the text.

2. The second new misconception is that the JOIN condition of the query will create a unique new column. As such, no alias is required in the SELECT clause: "there is only one column called sName so the table doenst need to be specified" - Participant 158. This misconception was held by three participants.

3. Finally, one participant wrote that not using an alias will remove the duplicate entries within a column: "disticnt, so c.cID and s.cID would find them all, and cID would remove duplecites" - Participant 248.

6.4.7 **Using DISTINCT on primary keys (PK_DISTINCT)**

Definition: Students believe DISTINCT is required to retrieve unique elements, even if the attribute is a primary key.

As adding DISTINCT on a primary key does not lead to an incorrect query, but merely complicates it, in our question in this study we ask for the *shortest* correct query, making the answers containing DISTINCT explicitly incorrect. Our prevalence scores show that the misconception answer was chosen 119 times over the two questions for 24% of the answers, with students being very certain about their answers (with an average score of 5.98 out of 7).

Forty participants say that DISTINCT is required to remove duplicates. It seems that the main reason that students use DISTINCT is that this is a rule they have learned. If the question mentions unique answers, then they apply DISTINCT. This is illustrated by the following concise explanation of participant 75: "unique = distinct". But even when students show to have a bit more insight into the workings of SQL, this template seems to hold up. As participant 236 writes: "needs second query in DBMS with bag semantics.", where the second query is the one including DISTINCT.

In our participants' elaborations, we also find a strong reflection of a lack of knowledge of primary keys. Seven participants wrote explanations that amount to the conclusion that IDs are not unique. For example, participant 11 wrote:

> "Query without select would just return all the cID's in the customer table which could be repetitions."

And participant 275 wrote:

> "Distinct gets only once the store id and not the duplicates"

Related to these primary keys is the idea that SQL is not set-based (mentioned by three participants), and thus may have duplicates. As participant 292 wrote: "It depends on whether store is a set or a multi-set" A multi-set, or bag, is a variant of a set that allows for duplicates. However, from the database schema, the participant could have seen that store IDs are primary keys and thus do not contain duplicates.

Finally, there was one noteworthy explanation of why DISTINCT was chosen, that we think illustrates a *new misconception*. It hints at a relation between DISTINCT and the effects of a Cartesian product. The explanation, by participant 31, is as follows: "I think if you don't mention distinct it combines all sIDs with all store names." It seems that this participant confused DISTINCT with the JOIN conditions that should be included in the WHERE clause. This answer warrants further investigation into the understanding of DISTINCT.

6.4.8 Lack of knowledge of primary keys (UNDERSTANDING_PK)

Definition: Students do not understand the implication of the presence of a primary key. Misunderstandings of primary keys were relatively rare for this question in our study. Only 5% of the answers could be considered misconceptions. However, there were many errors in question 2, in which students suggested that using GROUP BY, HAVING and COUNT was better than using a plain query (see Query 6.4). So, it seems that this query uncovered some other bias that we were not measuring.

For this question, there was only one student who gave a question-specific reason for their answer, and this seems to be a *new misconception*. The student confused the concept of the primary key with that of a JOIN condition. For question 1, participant 261 writes: "the names should be the same thus one of them should be part of the primary key".

The question for this misconception was:
Which of the following queries is the shortest correct query to retrieve a list of names of the customers who have at least one transaction?

The intended answer was:

```
SELECT c.cName
FROM customer c, transaction t
WHERE c.cID = t.cID
```

The answer that most students chose was:

```
SELECT c.cName
FROM customer c, transaction t
WHERE c.cID = t.cID
GROUP BY c.cName
HAVING COUNT(tID) >= 1
```

Query 6.4: Question two for UNDERSTANDING_PK, with the intended answer and the erroneous answer many participants chose.

6.4.9 (Not) equating primary keys (EQ_PK)

Definition: Students do not understand when they need to add JOIN conditions and comparisons, and when they do not. For this category, we measured their tendency to compare a primary key and foreign key when it was not necessary.

Our prevalence scores show that the misconception answer was chosen 114 times over the two questions for 23% of the answers, with students being relatively certain about their answers (with an average score of 5.25 out of 7).

The most common reasoning for our participants adding an exclusion is *we need to make sure the items are not the same* (21 times). This template reasoning is applied by participant 279:

> "considering pID is the primary key, we definitely need to check it, but we should also check the pName, as 2 products with different id's can have the same name apparently"

Indeed, in the question we are asking to return all different product names. As the participant mentions, indeed it could be the case that products with differing IDs have the same name. This is why they need to check for different names, and why they do not need to check for different IDs (as they are unique).

Related to this misconception is that *products with a different ID have a different name.* This was believed by at least ten participants and reflects a lack of understanding of primary keys. If one believes this, it is convenient to only check the IDs and ignore the names, as participant 272 is expresses: "The second one is the correct option because if the products have different ID's they will definitely have a different name. This is because pID is the primary key of product."

Finally, there were many participants who thought that IDs were not unique. We saw this already in subsection 6.4.7, but it seems a deep-rooted issue. For the questions in this category, it meant that the participants believed that products with a different name may have the same ID (17 instances). Participant 268 writes as explanation:

> "we want to make sure the pName is not the same but also the pID is not the same otherwise pairs of the same pID are always returned"

6.4.10 Alias behind the attribute (ALIAS_SYNTAX)

Definition: Students believe a syntax of attribute.alias is correct.

The misconception answer for this question was given only four times and is thus relatively rare. This might, in part, be due to the nature of multiple-choice answers, where the alternative answers are shown, which might trigger recognition.

Of the four misconception answers, one of these participants seems to have a previously undescribed, *new misconception*. Participant 280 writes:

> "All three options (a, b, and c) are correct because they all represent the column city, which is used to filter customers who live in Helsinki. The first two options (city

> and city.c) directly refer to the column city, while the third option (c.city) refers to the column city belonging to the table customer, referred to as c."

It seems that this participant has different interpretations of the meaning of c, depending on the location within the query. So, if the c is placed in front of the attribute, it accesses the aliased table (which is the standard behavior). But, if c is omitted or placed after the attribute, the "normal" column is called. This potentially refers to the attribute in the result table, which leads to circular reasoning that the participant might not have been able to foresee.

6.4.11 Only one copy of a table is required to compare within this table (SCOPING_SELFJOIN)

Definition: Students believe that comparing between rows can be done using only one copy of the table.

The prevalence table shows that this misconception occurred in 14% of our participants' answers, for 70 misconception answers. We identify three different question-specific reasonings. The first is that students think that any question that mentions counting elements requires the COUNT keyword. They have this template accessible and reach for it for every question, without checking whether the remainder of the query is correct. Four participants made arguments that *the COUNT keyword is required in order to count*; Participant 85 wrote that:

> "only one with count in it aand wee need at least 2"

and participant 196 writes:

> "we want at least 2 purchases, so we use count to determine the number of purchases."

Then there is another group of participants who also talk about counting, but are not explicit about needing the COUNT keyword. We coded these under the theme *COUNT can be applied in the WHERE clause*. Example quotes include "we can use count pid >2 and compare pid different to know there are at least 2 items in one transcation" by participant 101 and "count(pID) evaluates distinct type of products" by participant 236.

Finally, three participants suggest that *we can compare data within rows*. For example, participant 14 reasons about the answers:

> "Count means a customer has 2 houses in 2 different cities. Another one also checks for street. These are both wrong, I think we only have to fetch one table customers and in that table, we can compare if c.city != c.city"

These participants believe that only one copy of the customer table is required to be able to find participants who live in different cities. The elaborations do not give us further insight into why they would believe this.

6.4.12 Missing JOIN conditions in the WHERE clause (MISSING_JOIN)

Definition: Students do not understand that JOIN conditions are required to retrieve the appropriate results, retrieving a Cartesian product instead.

This pair of questions aimed to identify students (not) using JOIN conditions. Unfortunately, despite our careful checking of the questions, it turned out there were ambiguities in question one. On top of that, students' explanations for this question mentioned that it was too long and too difficult to understand properly on a laptop screen. Although this was not reported by students in our pilot, it may have led to guessing and low certainty scores for this pair of questions. Although the prevalence of misconceptions on these questions seems high, we should disregard their results.

As to the question of how there was still a problem after all questions were checked by teacher and students during the pilot, this is potentially due to evaluating some questions without utilizing the schema. Question 1 for MISSING_JOIN had a redundant table in it, which was filtered out again by the remainder of the query. However, we assumed during construction that this table was not redundant, and thus required a JOIN condition. Some students correctly identified that this was not necessary, landing them in the misconception category.

6.5 New misconceptions

Through the open elaboration answers, we also identified ten new misconceptions:

1. BRACKETS The WHERE clause requires brackets when it is part of a CTE.
2. SCOPING_WITH CTEs are not relations and thus cannot be included in the FROM clause.
3. DISTINCT The structure `attribute = (subquery)` allows the main query to take the first item from the subquery's result table.
4. DISTINCT The %-sign allows us to use the current date.
5. MISSING_ALIAS The JOIN operation removes ambiguity such that aliases are not required.
6. MISSING_ALIAS The JOIN operation creates a new column in the data.
7. MISSING_ALIAS Not using an alias means the rows in the result table are distinct.
8. UNDERSTANDING_PK Mixing up JOIN conditions and primary keys.
9. PK_DISTINCT Not using DISTINCT leads to functionality equivalent to a Cartesian product.
10. ALIAS_SYNTAX Placement of the alias (illegal placement is also considered) determines which column is accessed.

6.6 Discussion

In this section, we discuss our results and connect them back to our research questions.

First, we will compare our findings to those by Miedema et al. [111] that our questions were based on. We were able to find student answers reflecting the source misconceptions for nine out of twelve categories.

For BRACKETS, Miedema et al. found that students used brackets as a crutch [111, 114], and the misconception that students think the WHERE clause requires brackets. We find four participants in our population who believe this too. For NEQ, Miedema et al. [111] reason that this misconception is due to previous course knowledge, either from mathematics or studying Relational Algebra. Fifteen participants in our population mentioned that all answer options signaled inequality, and one felt that ≠ was more likely to be valid syntax than <>. For IS, Miedema et al. [111] state that the usage of IS (NOT) is a language-based misconception, inferred as correct from the natural language use of this word. This is also reflected in the reasoning of 38 of our participants, especially of those who responded that IS felt familiar. For SCOPING_WITH, Miedema et al. [111] make note that their students felt that defining the CTE was sufficient to be able to use it. This is reflected by our participants arguing that adding the name of the CTE in the FROM clause could be seen as unnecessary (4), redundant (14), or even illegal (8). For MISSING_ALIAS, we find that 13% of our participants struggle with this concept, in our case in combination with a JOIN. These struggles with aliases have also been exposed by Miedema et al., who identified inconsistent use of aliases, complications using aliases, and scoping mistakes with regard to aliases [111]. Finally, on EQ_PK, we find that 21 participants say that "we need to make sure the items are not the same". In their results, Miedema et al. [111] also mention students applying a query template in unfitting cases.

There were also three categories that we were not able to confirm. These are ALIAS_SYNTAX, where we found other reasonings than Miedema et al. [111], MISSING_JOIN due to ambiguity in our questions, and UNDERSTANDING_PK because of low misconception prevalence.

A noteworthy observation about UNDERSTANDING_PK is that there were many submissions with errors. Especially for the second query, many participants selected the most convoluted answer for a question asking for the shortest *correct* answer. As Query 6.4 shows, the question involves a COUNT keyword. One possible explanation is that the participants' template of counting was activated by the formulation of the question plus the presence of the answer, leading the participants to miss the correct answer. We confirmed this template-based thinking, first mentioned by Miedema et al. [111] in the questions SCOPING_SELFJOIN as well as EQ_PK, which also use the COUNT keyword.

Finally, some of the errors that we identified require deeper insights which were not provided by our participants in their open-text responses. One important one is the relation between DISTINCT and the Cartesian product, which was deemed confusing by one participant. Another is to explore the characterization of CTEs, as some students believe that they are not relations. Finally, students are trying to find pairs of data using only one table, whereas they should use a self-join. We leave these directions for future research.

6.6.1 **Implications for practice**

This investigation provides quantitative evidence on the prevalence of SQL misconceptions and highlights the ones responsible for the biggest obstacles in learning SQL and their interactions. We believe that the implications of our results pertain to education practice, education research, and language development.

In education practice, the results highlight the concepts that require special attention in instruction. Educators might not be accurate in predicting the frequency of mistakes, as has been investigated in programming [27], but to improve upon SQL instruction, we first need to identify where possible problems in knowledge transfer occur [163]. The most common misconceptions that we identified were IS, PK_DISTINCT, and SCOPING_WITH, all three having to do with data relationships and table joins. For these concepts, one of the ways to support instruction and knowledge transfer could be the use of tools that visualize data relationships or intermediate query results such as [33, 59, 100, 112].

Research is required to address faulty knowledge refinement and reorganization for the misconceptions that this study identifies as most common. Techniques for preventing or mitigating misconceptions are under-explored for SQL. One promising direction could be SQL-specific notional machines - pedagogic devices that support explaining and understanding complex concepts through representations (such as visualizations or tool-supported representations of program executions) or analogies (such as concept metaphors, for example, a programming variable as a label or a box) [57]. Refutation texts should be researched for mitigating misconceptions; they should consist of either two or three elements: a misconception, an explanation of the correct concept, and (optionally) a cue, which helps the student understand that the misconception is incorrect [185].

In terms of SQL language development, this study provides quantitative evidence for the ongoing discussion on SQL notation, its syntax, and its internal consistency. Several aspects of the SQL syntax are already being considered counter-intuitive [48, 105, 157].

6.6.2 **Threats to validity**

A threat to the external validity of our study concerns the students who answered the questionnaire, who might not be representative of all students of databases courses. This holds especially because misconception prevalence could be affected by the SQL instruction process, materials, and style. To mitigate this, we recruited participants from different universities across the globe. More data could give further insights, which is why the MSMI1 is publicly available.

Concerning construct validity, respondents potentially misinterpret or guess their answers to some questions. To mitigate this, we included several safety measures, including the pilot test and the certainty of response indicator, the results of which we took into account in the analysis. Unfortunately, our pilot did not catch the two pairs of questions that turned out to be independent, SCOPING_SELFJOIN and MISSING_JOIN. This is because the pilot was meant to capture errors in individual questions, and our sample size there was not large enough to calculate question dependence. As a result of the independence, we left these pairs out of our final published MSMI1 questionnaire.

II

Part 2 - Supporting students

7

Supporting Students with Querying Tools

On top of the research line in Part 1 of this thesis, which may lead to theory-backed SQL education in the long term, we also wanted to work on some more immediate interventions. As we learned in chapter 2, high cognitive load may be one of the reasons students struggle with SQL. Remembering the schema, the partial translations, and the question itself can be tough. Our idea was to create tooling that can help our students write queries, without hiding syntax. We developed SQLVis, a visual query representation that translates the SQL query into a graph-based representation. Our aim with SQLVis is to help students in learning how to write correct and portable SQL queries. Our evaluations show that SQLVis provides value to learners.

The contents of this chapter have previously appeared in **Miedema, D., and Fletcher, G.** SQLVis: Visual Query Representations for Supporting SQL Learners. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (Virtual Event, USA, 2021), IEEE Computer Society, pp. 1–9 [112].

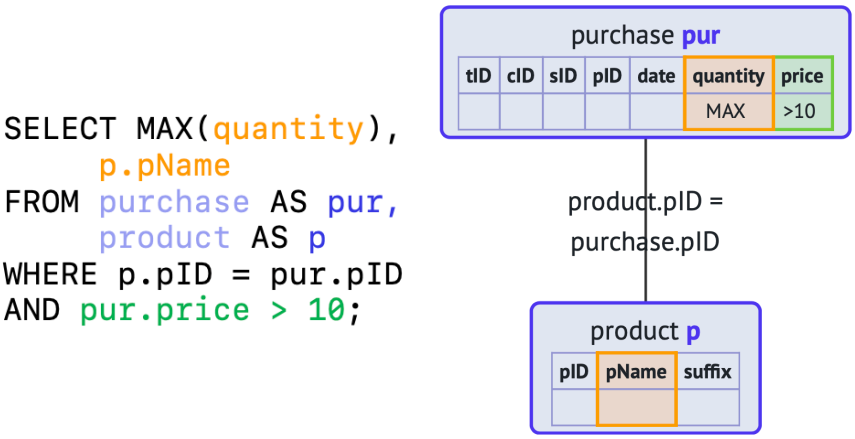


Figure 7.1: A user query and its corresponding SQLVis visualization. Tables are encoded as nodes, and constraints that link tables together as edges. The returned attributes are highlighted in orange, column-based contraints in green.

7.1 Introduction

Data is commonly stored in relational databases. The most popular language through which this data can be queried is the Structured Query Language (SQL). Previous research has shown that using SQL is difficult and error-prone [3, 37, 87]. Extensive training is required before one can use SQL effectively. In addition, the cognitive load of writing queries is high [16, 119]. Various aspects may play a role, for example: keeping the database schema in working memory, the system only catching certain types of errors, low expressive ease, and losing track of your query while writing it [38, 119, 129]. These issues occur especially for novices, which hinders the process of learning SQL.

Despite the high barrier to entry, SQL is ubiquitous [180], in education and practice. To simplify the querying process, many researchers have presented methods that allow for visual query building, such that no knowledge of syntax is required for writing queries in SQL [88, 126, 168, 184]. Such Visual Querying Systems (VQS) generate a query from a visual representation constructed by the user. A VQS simplifies interactions with the database by abstracting away from the language syntax, which leads to increased information throughput [32]. However, they also increase system dependency, as a VQS obfuscates syntax instead of supporting the learning of SQL.

Another approach to using visualizations to support query formulation is to generate a representation from a written query. This can either be done by annotating the result table, or in the form of a separate visualization. We call the former Visual Result table Representations (VRR), and various such systems exist [33, 50, 91, 123]. However, this approach does not scale well for queries involving many (or large) tables, or queries with many results. In contrast, a Visual Query Representation (VQR) is a visualization that represents the *query*, without taking into account the result table. This has the advantage that the representation is relatively small and more structured than a representation on the result table. Figure 7.2 contains an overview of the three types of visual tools.

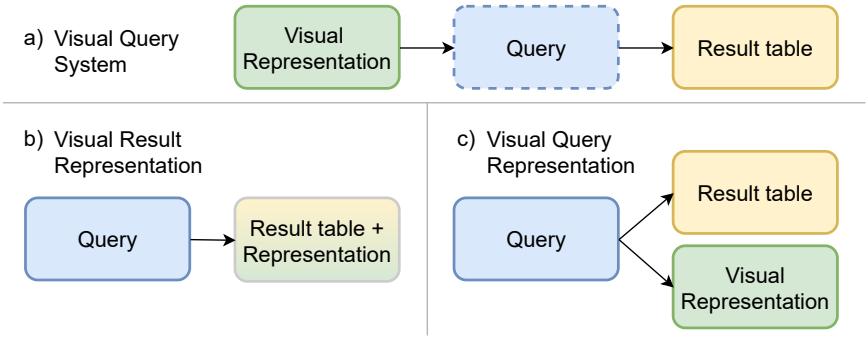


Figure 7.2: A visual comparison of VQS, VRR, and VQR.

The VQR approach of query formulation and post-hoc visualization facilitates the learning of the query language. In addition, a VQR supports the learner by visualizing the meanings of SQL commands, keeping track of query formulation, and relieving the user of the burden of keeping the database schema in memory. Juxtaposing the representation and the query can help the user understand the effect of using various SQL clauses. So, contrary to the VQS approach, with a VQR we can support the learner, while also familiarizing them with SQL. For learners, a VQR is also preferable over a VRR as the former is a more direct and compact representation. Through all these advantages, a VQR can be a great aid in database education and training.

VQR is a relatively new and novel approach, representing a paradigm shift from user to learner. To our knowledge, QueryVis is the only prior VQR work [100]. In contrast to supporting learners during query formulation, however, QueryVis focuses on the interpretation of existing queries. This leads to a difference in design decisions that we present in section 7.3.

Our contributions. In this chapter we present SQLVis[1], a Visual Query Representation for SQL learners. SQLVis leverages a graph-based query representation. A graph structure for VQR increases understanding by adding information, as SQL's implicit relations in the data (through foreign keys) can be confusing. SQLVis makes these foreign key relations explicit, which lowers the cognitive load.

We proceed first with grounding the design of SQLVis in the literature on SQL- and graph visualization. When then analyze SQLVis with respect to two research questions:

Q1 How do users assess SQLVis?

Q2 What is the effect of using SQLVis on the query formulation process?

We evaluate **Q1** and **Q2** with a qualitative and a quantitative study, which show that SQLVis is evaluated positively and query formulation through SQLVis leads to fewer mistakes than query formulation without SQLVis. We conclude with pointers for future research building upon our contributions.

[1] `https://github.com/Giraphne/sqlvis`

7.2 Related work

7.2.1 Visual representations and cognitive load.

In this section, we give an overview of reasons why adding a visual representation to a system can be beneficial. First of all, a visual representation provides the opportunity to apply parallel processing. This is possible because short-term memory gathers information through two separate channels within working memory: the visuo-spatial sketchpad and the phonological loop. Both these information channels and the working memory as a whole have a limited capacity. The amount of working memory resources in use is called the cognitive load. The concept that the chosen learning method influences cognitive load was posed by Sweller [170].

Queries in SQL are purely textual, and thus processed in the phonological loop. By adding a visual representation in the interface, some load can be taken off the phonological loop by redistributing it to the visuo-spatial sketchpad. This dual-coding of information has a memory-enhancing effect through the freeing of resources [45, 83, 154]. This holds especially in cases where the textual and visual data are highly related [82], close together, and present new information [154].

There are various ways in which a representation can support data analysis: through parallel processing, as external memory, and by imposing structure on the data [186]. Yung et al. found that a visual representation helped their participants focus their attention on the most essential elements [203].

Visual representations can also aid the development of appropriate mental models. Schnotz et al. found evidence of a correspondence between the structure of a graphical representation and its associated mental model [154]. There also seems to be an effect of experience: as their knowledge is typically fragmented [45] novices benefit from pictures in text [154]. Much of the information presented to an expert is already in their long-term memory and thus will not overwhelm them [45]. However, incorrect pictures may confuse experts, thus a representation needs to be appropriately designed [154]. The integration of new information with existing knowledge is an essential part of meaningful learning [107].

7.2.2 Representing SQL queries.

As research has shown that one of the main problems in query formulation is high cognitive load [16, 119], SQL can profit from this effect of visual representations. Indeed, recent work has underscored the importance of further research on visual support for query formulation [60]. High cognitive load leads to errors in query formulation, with specific SQL keywords that users make the most mistakes in. Examples of difficulty-inducing concepts are aggregate functions such as AVG, COUNT and MAX [26, 180], grouping constructs such as GROUP BY and HAVING [3, 26, 91, 119, 180], and concepts such as subqueries and negation [108, 180].

Subqueries can be difficult to understand due to their intrinsic nested nature. Queries can contain many levels of nesting, making them hard to write, and even harder to understand. For the visual representation of subqueries, there is a clear-cut answer: delimiting each level

of nesting and each subquery within that with a box, nesting the boxes as the queries are nested. This is supported by earlier research, such as [183]. Another method is to represent nesting through directed arrows, which indicate a reading order [61].

There is less consensus on the optimal way to represent negation. It is relatively easy to show the presence of a concept, but how does one show something that must not be there? Earlier work suggests the use of complementary colors, distinguishing between existing and not existing [93]. Others suggest different borders for different types of complex operations (including conjunction and disjunction) [21, 61]. The most primitive form of negation through colors and borders is the notion of a 'cut', as introduced by Peirce in the 19th century [39]. Finally, negation can also be expressed through the use of symbols such as $\neg$ or $\sim$ [39].

There are various aggregate functions available in SQL. All correspond to mathematical functions that can be evaluated on the data. They are always called on one or more columns in the data, and thus it would make sense to highlight them there. Thalheim developed Visual SQL [183], a representation of SQL as boxes and connections. They define aggregation elements as algebraic expressions that are added to the unit on which the expression is applied. Similarly, Leventidis et al present aggregates in the column of the schema that they are applied on [100], similar to how they appear within the query.

Of all the SQL concepts, GROUP BY has been shown to be one of the most difficult. As with aggregation, GROUP BY is usually called on one (or more) columns within a table. We therefore can apply the same principle here as before, either the attribute can be highlighted, or we can add text. Where Leventidis et al. chose to go with textual representations for aggregation, for GROUP BY they go with a grey highlight on the attribute [100]. In SAVI, Cembalo et al. also chose to display GROUP BY as a highlight on the column they group by [33]. Thalheim mentions that operations, such as GROUP BY, are added to the units, but do not show an example [183].

Besides GROUP BY, JOIN is one of the concepts that students make the most mistakes with. Joining tables is a very abstract concept, as it creates relations between tables that are only implicitly available in the data. A visual representation of this concept can be beneficial, as the explicit drawing of lines, or edges, shows in detail what is implicit in the JOIN condition. These edges also present us with a medium for displaying comparisons between attributes.

One of the most traditional forms of graphs to represent data was Conceptual Graphs [167], which were used to represent database schemas. Conceptual Graphs can be used as graph-based knowledge representations, translating text into graphs, which is similar to our application of SQLVis as a graph translation of the SQL query.

Another system that is similar to SQLVis is QueryVis. QueryVis is a system that simplifies the interpretation of SQL queries by generating visual representations [100]. Their representations are also graph-based, in the sense that each table in the query becomes a separate entity, and there are lines connecting them. The focus of their work is on query interpretation (of existing queries), rather than on query composition. Our study focuses on query formulation, and the use of visual representations to support this.

As discussed in the Introduction, various Visual Querying Systems exist. These VQSs allow users to avoid writing queries textually, instead using visual representations or highlighting

data. They thus do not focus on representing the queries themselves. Besides VQS, some Visual Querying Languages exist. One example of a visual querying *language* is QBE [206] which allows for querying databases through visual tables. SQLVis has similarities to QBE in the visual sense, but there are two significant differences. First, our representation is styled as a graph, highlighting the implicit connections between tables in a query by drawing them. Second, our representation is intended as an educational support mechanism and thus requires the users to write their own SQL queries instead of generating the query from the visual representation.

7.2.3 **Graph visualization principles.**

As SQLVis is a graph-based VQR, the design is informed by research on graph visualization.

The leading theory behind drawing static graphs are the Gestalt principles, which represent basic elements of perception, e.g., elements with common features (such as color) are often perceived as groups. Bennett et al. [15] distinguished between principles for combination (similarity, continuation, proximity) and principles for segregation (symmetry, orientation). This use of rules guiding perception can aid in the drawing of graphs as they present good aesthetics [15]. Ware et al. [192] also focus on Gestalt principles in graph drawing. They promote the principle of good continuation: paths in graphs are better perceived if the nodes along the path form a smooth continuous sequence and, furthermore, paths can be emphasized if the line connecting them is curved [192].

In addition to Gestalt principles, graph visualization heuristics have been proposed. These can be divided into categories for node placement, edge placement, and graph layout; furthermore, domain-specific heuristics can be applied [15]. Examples of node placement include an even distribution of nodes and the clustering of related nodes, as well as the maximization of node orthogonality [15]. Spatial alignment within a graph is important as it is one of the ways in which cognitive load can be reduced [107]. Edge placement recommends that edges should cross as little as possible, and maximum edge length should be minimized while minimum edge length should be maximized [15]. In addition, edge bends should be uniform. There are various types of graph layout algorithms, including force-directed layouts, attribute-based layouts, and constraint-based layouts [62]. A subset of constraint-based layouts are the hierarchical layouts, of which Sugiyama's is the most well-known. A hierarchical layout fits SQL queries well, as there is a hierarchy through the presence of subqueries. Additionally, consistent layout is important because it can preserve the mental map of the viewer [140]. This means that the location of a node should stay the same throughout subsequent graphs, supporting identification as the same node.

Interaction can also be a way to increase graph understanding. One idea is that exploration may reveal insights that were hidden in a static image [186]. Beck et al. [13] argue that interaction with a graph can compensate for worse graph design concerning visualization criteria. In the end, users are essential to the visualization process; Tory et al. [186] therefore argue for the application of human factors and a user-centered design philosophy to be adopted in visualization design.

7.3 Design

Building on the research literature, we designed SQLVis. In this section, we explain all aspects of SQLVis in light of the research presented in subsection 7.2.2 and subsection 7.2.3. Note that SQLVis is a research prototype and therefore incomplete. SQLVis handles all queries that students typically pose in their homework, but lacks functionality for more complex actions, such as view expansion, indexes, and insert and delete queries.

Firstly, SQLVis is graph-based and explicitly shows the relations between the tables that are only implicit in the SQL query itself. All tables are represented as nodes, and all relations are represented as edges. The nodes include the name of the table they refer to, plus an alias in bold if that was defined by the user (see Figure 7.1 and Figure 7.4b). SQLVis also allows for interaction with the visualization to increase exploration, as suggested by Tory et al. [186] and Beck et al. [13]. Users of SQLVis can move the representation around, and expand all of the nodes to show the table's schema. Our decision to display the schema horizontally, instead of vertically as other authors have done, gives us more space to add additional content in the tables.

SQLVis highlights any interactions of the query with the table in green and orange (see Figure 7.3). This calls attention to all elements from the SELECT and WHERE clause, including any comparison values that are present in the query. These highlights for SELECT are different from the approach by Leventidis et al. in QueryVis [100], who chose to create a separate view that includes everything in the SELECT clause. SQLVis does not include such a view, as we do not focus on the result table of the query, but rather on query formulation.

All complex SQL queries contain subqueries and other types of nesting. To visualize these subqueries in the most intuitive way, SQLVis draws boxes around these subqueries as suggested by Thalheim [183]. To distinguish between different subqueries on the same level, and nested subqueries on different levels, SQLVis draws each level of nesting in a different saturation (see Figure 7.4b). The use of these different colors helps to give an immediate overview of the level of nesting in the query. In case a subquery is negated, for example by using NOT EXISTS or NOT IN, SQLVis displays this negation in words. Other options, such as a different background color or a border for the box led to a very cluttered representation.

Negation may also occur on conditions in the WHERE clause. In that case, SQLVis takes the != or <> from the query and displays it in the schema with the text or numbers of the condition, such as !=London in Figure 7.3.

Other elements that cause problems in query formulation are aggregation and GROUP BY. For these two types of actions, SQLVis highlights them in the column where they occur. This approach is similar to the suggestions by Thalheim [183] and Leventidis et al. [100]. As there may be more than one column used for grouping, SQLVis also includes a number that indicates the grouping order. For an example, see Figure 7.4a.

As mentioned before, SQLVis applies a graph-based design to explicitly show any implicit relations. This holds specifically for JOIN conditions, both those using the ON keyword and the implicit JOIN conditions in the WHERE clause. If the JOIN keyword is used in the query, the JOIN type is also displayed on the edge that contains the JOIN condition.

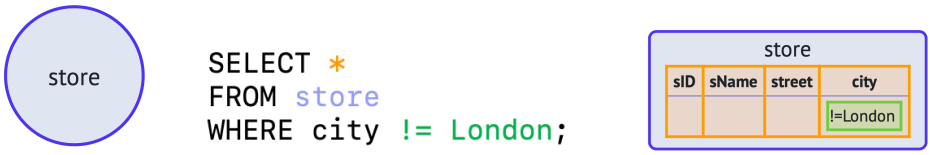


Figure 7.3: SQLVis representation for a simple query (collapsed on the left, expanded on the right).

For the graph visualization principles, we apply as many Gestalt principles of grouping in SQLVis as practically possible. After all, each graph represents a single query and thus should be seen as a single entity. Nodes are drawn close together and look the same in shape, size, and color. The SQLVis representation is drawn as symmetrical as possible, with subqueries of the same depth drawn on the same level in the graph. This also implies spatial alignment to reduce cognitive load, as suggested by Mayer et al. [107].

Finally, we reflect on the concept of consistent layout as explored by Purchase [140]. This is not applicable in SQLVis, as there is no need to identify nodes: they are identified automatically because they have their name on the face of the node. Furthermore, subsequent questions consider different tables in the database. SQLVis is consistent in the sense that the same query leads to the same layout, it depends on the order in which the tables are called in the query.

Note that SQLVis does not support syntax errors: in such cases no AST can be parsed, which is required for building the VQR. While visually highlighting syntax errors is straightforward, the independent research question of how Visual Representations can best assist students when syntax errors occur is an interesting challenge outside of the scope of this chapter.

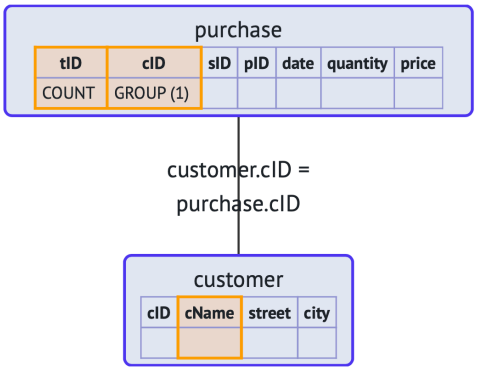
7.4 Qualitative study - Methodology

With this first study, we aim to answer item **Q1**: How do users assess SQLVis? To gather feedback on the potential value of the system, we ran a user study in two parts: 20 minutes of query formulation with the support of SQLVis and an interview to discuss the design.

Participants. There were five participants with moderate knowledge of SQL. All participants were male with ages between 20 and 28. Their self-reported skill level was 6 to 8 on a scale from 1-10. The study took 40 minutes and participants were compensated seven euros for participation.

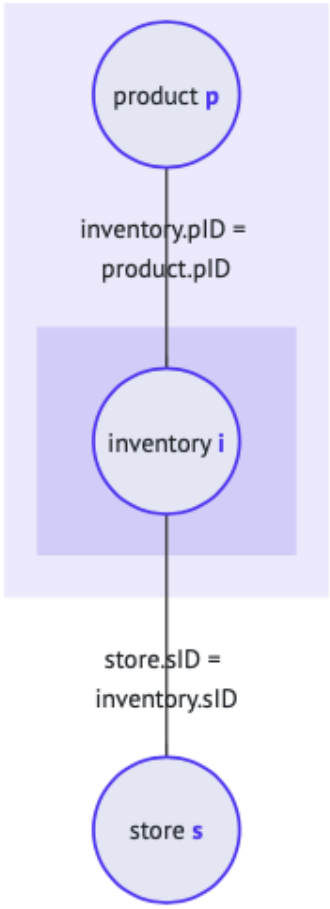
Design. We designed a three-part study. First, we need our participants to experience the system. For usage, we presented them with four query formulation problems and had them use a system with integrated VQR to solve these problems. We decided not to use a think-aloud method, as writing queries is high in cognitive load, and talking could disturb our users. Therefore, we had them work undisturbed and discussed SQLVis afterward in a semi-structured interview.

Procedure. At the start of the study, the participants were welcomed and signed an informed consent form. We then presented them with the workings of the query formulation system and answered any questions they might have. They sat down with a laptop to work on several query formulation problems for approximately 25 minutes. See Table 7.1 for the questions




```
SELECT customer.cID,
customer.cName,
COUNT(purchase.tID)
FROM purchase, customer
WHERE customer.cID =
purchase.cID
GROUP BY customer.cID
```

(a) A query to count the number of purchases per customer




```
SELECT s.sID, s.sName
FROM store AS s
WHERE NOT EXISTS (
  SELECT p.pID
  FROM product AS p
  WHERE NOT EXISTS (
    SELECT *
    FROM inventory AS i
    WHERE s.sID = i.sID
    AND i.pID = p.pID))
```

(b) A query to find stores with all items in stock.

Figure 7.4: Two representations of more elaborate queries, containing groupby, aggregation, and nested subqueries.

	Questions
1	List all the product IDs of products that were bought by at most two different customers.
2	List the customers (ID and name) who purchased on the same date both a product with name 'Onions' and a product with the name 'Coffee'.
3	List the IDs of customers that made a purchase at every store.
4	Find the name of the product that is sold for the highest price (ever) and the name of the store that sold it for that price.

Table 7.1: The query formulation problems for the participants of the qualitative study.

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 7.2: The database schema, primary keys underlined.

and the corresponding database schema in Table 7.2. Their interactions with the system were screen-recorded. After getting to know the system, we finished with interview questions about the system, and any other questions or discussion points the participants may have had.

Ethics. The participants were not financially compensated for participation in the study. The study design and data collection were approved by the Ethical Review Board of Eindhoven University of Technology.

7.5 Qualitative study - Results

In this section, we focus on the semi-structured interview. In the interview, we asked the participants what their general impression of the VQR was. Below we discuss some quotes from our participants.

> "[SQLVis] is a good idea. It is more organized than looking at tables, you can see exactly the parts you need instead of having to search through large tables." - Participant 1

Our first participant is happy with the structure that SQLVis brings. One advantage of the representation is that the nodes can be expanded to show their internal schema. This can help the user find the appropriate attributes to use within their query.

> "The design makes sense, for example using highlights to show which columns you are selecting." - Participant 2

The highlights are appreciated by our second participant. Besides the node-link structure

depicting the query's implicit relations, the highlights look at the specifics of the selections and conditions. This helps to find any mistakes in the query (for example using an incorrect table) and helps the user to finalize the query.

Participant 3 refers to the use of the VQR as a tool to keep track of the current state of the query; has the user incorporated all elements they wanted and thus completed their query?

> "I think [SQLVis] is a good idea, because you can often lose track of what exactly you are [writing], what columns you have selected etcetera. It helps with that." - Participant 3

During query formulation, users need to keep track of many aspects such as the database schema and which parts of the question they have already translated. This causes a high cognitive load. SQLVis shows all operators and operands of the query in its current state, thereby lowering the cognitive load associated with query formulation.

Participant 4 looks at this from another direction:

> "SQLVis was fun to see but I have not used it a lot. Mostly because the visualization only shows what you already have, which does not help you move forward." - Participant 4

Indeed, SQLVis will not assist the user by suggesting elements to include. As we aim to support learning, we encourage learners to formulate queries themselves rather than automating part of the query formulation process.

Participant 4 was able to finish all four query formulation problems. As the queries were easy for them, there was no need to use SQLVis as a support mechanism. For struggling students, however, the value can be quite high.

> "The VQR looks good but I didn't really use it. Mostly because I am not used to having such functionality. If I were to become more familiar with the system I think it could be helpful to find mistakes in your queries. Finding mistakes is difficult for complex queries." - Participant 5

Participant 5 highlighted the fact that as SQLVis was still unfamiliar to them, they did not use it a lot. However, from the introduction of it at the start of the study, they suggested that it would be useful for finding mistakes in the query.

Summary. Overall our participants were positive about the application of SQLVis as a support mechanism for query formulation. The design was described as sensible, clear, and fun. Possible applications mentioned by participants include finding mistakes and keeping track of query formulation. The qualitative study, however, was too short for participants to get used to incorporating it into the query formulation process.

7.6 Quantitative study - Methodology

In this second study, we aim to answer item **Q2**: What is the effect of using SQLVis on the query formulation process? To answer this question we asked Bachelor students taking our second-year databases course to write SQL queries through Jupyter Notebooks. We divided the students into two groups (a between-subjects protocol), using two different notebooks:

	Questions
query3_2	Select all distinct combinations of names and ids of customers who have both a shoppinglist and a purchase, both for the same date in 2018.
query3_3	Select all different names and ids of the customers who never made a purchase at a store with the name 'Kumar'. Also include the customers who never made a purchase, but are still represented in the customer relation of our database.
query3_4	Select all different names and ids of customers who made at least one purchase at a store with the name 'Kumar' and never a purchase at a store with a different name than 'Kumar'.
query4_3	Write a SQL query for the following RA query: $\Pi_{sName,city}(store) \div (\Pi_{city}(customer) \cup \Pi_{city}(store))$
query4_4	List the names of all of those customers, and only of those customers, that spent an amount of money on any one date that is at least 75% of the maximal amount of money ever spent by a single customer on a single day.
query4_5	Per city, how many customers living in the city have made at least one purchase from a store in London?

Table 7.3: The questions for the quantitative study. The first number in the question name refers to the week they were posed.

one plain (control group), and one that included SQLVis as a Python package. Our participants were divided into groups by last name. This random division over the groups should make sure that the groups are balanced in terms of skills and demographics.

Design. To compare performance in an AB test, we needed to find a way to have people work on the same problems in both conditions. The opportunity arose to use the authors' institution's second-year introductory Databases course. To analyze not only the final results but all intermediate attempts, we set up a Jupyter notebook. Students at the authors' institution are familiar with writing SQL queries through Jupyter notebooks at this point of the Bachelor program. This, in combination with the flexibility that Python offers, is the reason we went with notebooks. We created two similar notebooks, both including the homework questions, with one version including SQLVis. To invoke SQLVis, students can call its visualize() method, which uses the database schema and query text to generate a visual representation and displays this directly below the code cell that contains the visualize() call. Additionally, the notebooks logged all interactions of the student, including timestamps.

Materials. Notebooks were offered for two different homework assignments in weeks 3 and 4 of the course. The notebooks contained cells with a description of how to participate, the problem formulation for each of the problems, and room to formulate answers. Students could use the notebooks locally. Usage of the notebooks generated Python logs in separate files. For the non-visual notebook, there was just one cell available for each question, which allowed the participants to define and execute a query at the same time. For the visual notebooks (including SQLVis), there were three predefined types of cells: define, visualize, and execute. Execution of each of these cells is logged as an action of the corresponding named type.

The query formulation problems included in the notebooks are listed in Table 7.3. This study used the same database schema (Table 7.2) as the qualitative study described above.

	Non-visual	Visual
Participant count	27	16
Total interactions	3377	2624
Total executed	3377	921
Count correct executed	174	111
Count incorrect executed	2216	496
Count error executed	960	310

Table 7.4: Statistics on query execution by our participants.

Procedure. Students were offered the study's notebook through the University's Learning Management System (LMS). They were informed through the LMS that the notebooks were uploaded and received instructions on which variant of the notebooks to use and how to submit. Upon submission of their homework in the LMS, students were asked to also submit any logs they may have generated. These logs were then downloaded and organized.

Ethics. The participants were not financially compensated for participation in the study. The study design and data collection were approved by the Ethical Review Board of Eindhoven University of Technology.

7.7 Quantitative study - Results

In this section, we call the notebooks that include SQLVis 'visual', and the control condition 'non-visual'. Answers can be either correct (the result table of the attempt corresponds to a pre-defined correct result table), incorrect (the result tables do not correspond), or erroneous (the DBMS returns an error upon execution of the query).

First, we will present general numbers on participation in Table 7.4. The table shows that the groups were not equal in size. However, the number of interactions with the notebooks shows that the size of the data is adequate for analysis. As we are evaluating performance in this study, we look at the four different outcomes that a query could have: correct, incorrect, producing an error, or resulting in a time-out. For both groups, we see that incorrect answers are most common (65.6 and 53.9 percent), followed by errors (28.4 and 33.6 percent).

Correctness and attempts. To see how these correct answers are distributed, we analyze per query and per participant whether they found at least one correct answer for each question. The results can be found in Figure 7.5. The proportion of participants with a correct answer for the first question is approximately equal. Then for query3_3 until query4_3, the participants in the visual condition scored better. It is interesting to see that the majority of students were not able to find a correct answer for query4_4 and query4_5. If we compare this to the number of attempts made on each question, as shown in Table 7.5, we see a difference in behavior between the non-visual and visual notebooks. The participants with visual notebooks have more attempts on the first four questions (given that they have fewer attempts overall), and much fewer attempts than the non-visual participants for the last two questions. For the visual notebooks, there is an outlier on query4_3 specifically, where a lot of attempts were made. This is not due to any one participant, but an effect of an increase

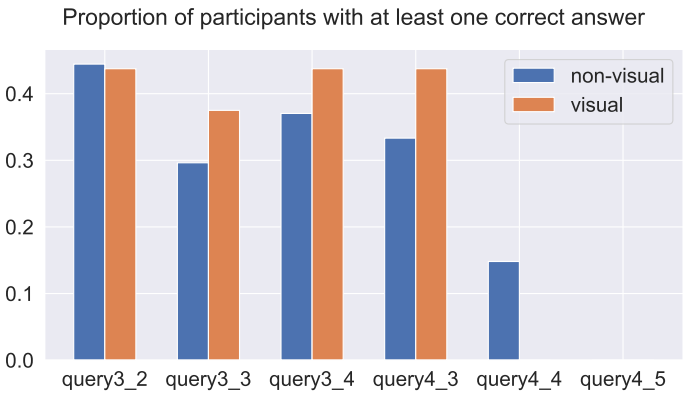


Figure 7.5: Counting the proportion of participants having at least one correct answer per question.

over multiple participants. In the other direction, query3_4 is an outlier with a low number of attempts in both conditions. This may be due to the fact that the question is closely related to the question before it. Participants could reuse parts of their query to answer the question more quickly.

Statistical testing. We hypothesize that students using the visual notebooks are more effective query formulators. This means more correct answers, fewer incorrect answers, and fewer errors. Because of the differing group sizes, we test this via proportion Z-tests. For the visual notebooks, we just include the 'execute' cells, as there is a duplicate-query effect for define cells (which are counted as correct or incorrect too).

For *correct* answers, we have a proportion of 12.1 percent for visual notebooks and a proportion of 5 percent for non-visual notebooks. The null hypothesis asserts equal means, but the calculation of a one-sided proportion z-test shows that the proportion of correct answers for the visual notebooks is significantly higher than that of the non-visual notebooks, with a p-value of 4.35e-14. For *incorrect* answers, we have a proportion of 53.8 percent for visual notebooks and a proportion of 65.6 percent for non-visual notebooks. The null hypothesis asserts equal means, but the calculation of a one-sided proportion z-test shows that the proportion of incorrect answers for the visual notebooks is significantly lower than that of the non-visual notebooks, with a p-value of 2.7e-11. For *error-producing* answers, we have a proportion of 33.6 percent for visual notebooks and a proportion of 28.5 percent for non-visual notebooks. The null hypothesis asserts equal means, but the calculation of a one-sided proportion z-test shows that the proportion of erroneous answers for the visual notebooks is significantly higher than for the non-visual notebooks, with a p-value of 0.001. The absence of advantage for the visual condition can be explained by the fact that the representation cannot be generated when the query contains a syntax error, as parsing the query's syntax tree fails in that case.

Higher efficiency can also be expressed in a lower number of attempts. We compare the means via a t-test. The null hypothesis asserts equal means. We take an alternative hypothesis that asserts lower means for visual notebooks. We find that we can reject the null hypothesis with a p-value of 0.01.

Query	q3_2	q3_3	q3_4	q4_3	q4_4	q4_5
Non-visual	460	587	299	686	696	649
Visual	149	142	101	242	147	140

Table 7.5: Number of attempts per question.

Finally, we can express efficiency in the time taken to solve the questions. We found that participants in the visual condition took 2 hours and 4 minutes on average, and participants in the non-visual condition on average took 2 hours and 25 minutes to finish the exercises. A t-test did not show a significant difference between the conditions.

Error analysis.[] In short, participants in the visual notebooks had fewer attempts, more correct answers, fewer incorrect answers, and more errors, than participants without the visual representation. To gain more insights into these errors, we undertook text processing on the errors returned by the database system, to find out what the most common types of errors were. In summary, the most common error is the plain syntax error, which represents 58.5 percent of errors in non-visual notebooks, and 33.5 percent of errors for visual notebooks. Other noteworthy errors that are made more in the non-visual notebooks are *incomplete output*: 3.75 versus 1.6 percent, and errors regarding the number of returned columns not matching the number of expected columns: 7 vs 4.5 percent. For the visual notebooks, an error that occurs more often is *ambiguous column name*: 5 percent for non-visual notebooks vs. 13.2 percent for visual notebooks. Participants in both conditions have similar counts for *no such column*: 15.4 versus 16.4 occurrences.

Although the visual representation did not work in cases of syntax errors, we can draw a parallel between the error types and the visual representation usage, as query definition is a process of refinement. For example, participants using the visual representation might have a hard time with correct column names as the visual representation extracts the correct ones from the schema automatically. From the lower incidence of general syntax errors, it seems that the participants in the visual condition were more careful in their query formulation and thus made more meaningful errors.

SQLVis usage. We define three different interactions in the visual notebook: define, visualize, and execute. Usage statistics show that define was called 1037 times, execute was called 921 times, and visualize was called 769 times. As the number of define calls is higher than the number of execute calls, it seems that the visualization helped in the refinement of query formulation. The fact that using the visual representation was optional, and our 16 participants chose to use it 769 times is another indication of usefulness.

To analyze SQLVis quantitatively, we define helper patterns:

1. Execute a query, visualize this query, define a new query.
2. Visualize a query, define a new query, visualize this query.
3. Execute a query, define a new query, visualize this query.
4. Define a query, visualize this query, define a new query.

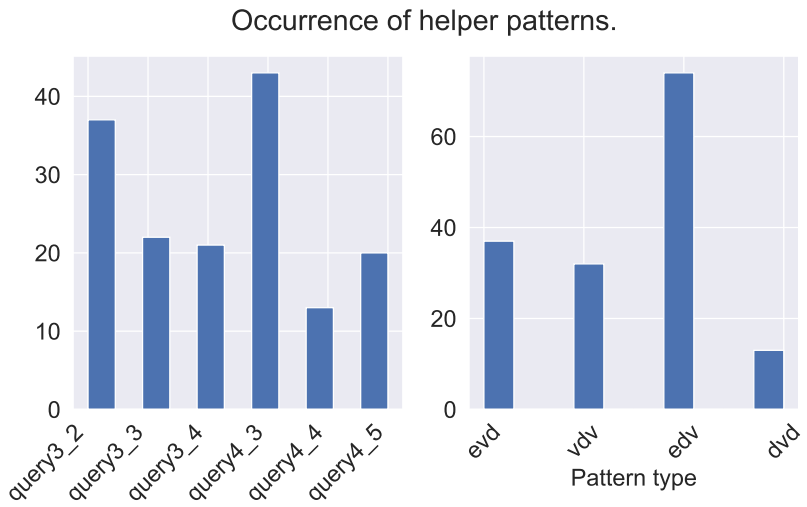


Figure 7.6: The division of helper patterns over the homework questions. The pattern types are identified by the first letter of each action, so evd represents execute, visualize, define.

These patterns highlight the cases in which the visual representation is used as a tool for repair, verification, or query building. For all patterns, we require that they are subsequent attempts on the same question, the initial query was incorrect, and they cannot include syntax errors as in those cases the visual representation can't be used for problem-solving purposes.

We first count the incidence of these helper patterns together: they occur 156 times in total. In Figure 7.6, they are divided over their corresponding question and counted by type. Helper pattern 1 was used most on query4_3 and query4_5, helper pattern 3 was used most on query4_3, and helper patterns 2 and 4 were used most on query3_2. Overall, the highest incidence of repair patterns occurs for query3_2 and query4_3. As query4_3 requires a translation from Relational Algebra to SQL, the addition of an intermediate representation might have been especially valuable for the participants. For the pattern types themselves, we see that pattern 3 is the most popular. This may be explained by the order provided in the notebook: define, visualize, execute, of which pattern 3 is a variant.

Another aspect of these helper patterns is the change in query complexity. We define complexity as the number of operators and operands in the query, an idea posed by Bowen et al. [24]. We took each instance of the helper patterns and calculated the difference in query complexity between the first and the last query in the pattern of three. The scores were then categorized by question and plotted in a box plot, see Figure 7.7. We see that for the questions in week 3, the median complexity difference is 0. For week 4, medians are above 0, with an average score of 0.5 to 3.5. This means that on average, at least one operator or operand was added to the query. It seems that in week 4, the helper patterns on average increased the complexity of the queries, which means participants were using SQLVis as query formulation support.

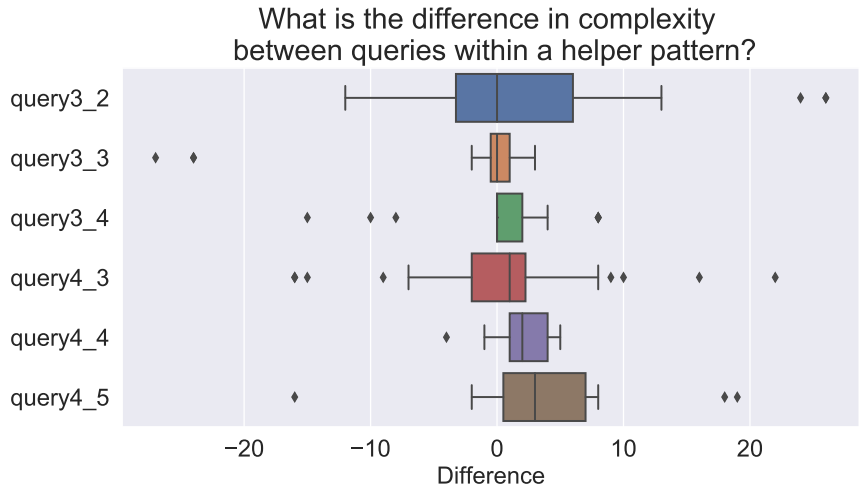


Figure 7.7: The usage of helper patterns leads to new queries. What is the difference in Bowen complexity score between the old and the new query?

Summary. Our participants actively used SQLVis, with 769 calls to the visualize command. Participants using SQLVis wrote significantly more correct queries than the control group and significantly fewer incorrect queries. SQLVis also facilitated efficiency, with a significantly lower number of attempts. Our definition of helper patterns showed that our participants used SQLVis as a tool for query formulation and verification.

7.8 Limitations

The first limitation of this work is the low participant count for both studies. Especially for the quantitative work, the statistics are less trustworthy because of the smaller sample. One factor that potentially mitigates this problem is the large number of interactions generated by the 43 participants. Furthermore, it seems unlikely that offering an optional extra resource has negative effects on the participants.

The second limitation is that the study was ran in a single university, with students being taught by a single lecturer and a small set of instructors. We are interested to learn whether students at other universities would have similar experiences. Therefore, SQLVis is openly available as a Python package for others to use in their classroom.

The third limitation is the self-selection bias. The students participating in the quantitative study potentially have different characteristics from the ones who did not participate. To make the difference between the students within and outside the study as small as possible, we provided the Jupyter notebooks for the study to all students, regardless of participation in the study. To participate in the study, the only thing required was for students to upload their logs to the Learning Management System in addition to their homework solutions. This made the hurdle for participating very low.

8

Understanding Student Engagement

Our final contribution in this dissertation is a second work on supporting students. Our idea was to research contextualization: providing a context or story instead of a more abstract problem statement. We know from educational research that contextualization of assignments can be beneficial: it may increase engagement and lower dropout rates. As database education is intrinsically contextualized through the data chosen for the schema, we aimed to investigate which aspects of these schemas students would find engaging. Our study centered around students in an advanced data systems course, who had to design and implement their own database, with the goal to make it engaging for a novice. We asked them to write reflection reports on why they felt their database was engaging. Our takeaways from this study can directly be applied by data systems lecturers to evaluate their example databases.

The contents of this chapter are based on **Miedema, D., Taipalus, T., and Aivaloglou, E.** Students' Perceptions on Engaging Database Domains and Structures. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto, Canada, 2023), pp. 122–128 [117] and **Taipalus, T., Miedema, D., and Aivaloglou, E.** Engaging Databases for Data Systems Education. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland, 2023), pp. 334–340 [177].

8.1 Introduction

Data management skills are becoming more and more central to the tasks of a computer scientist. Data management is of high importance in institutions using large amounts of data, for organizing data distribution and decentralized query execution. It is also core to the area of artificial intelligence, as no model can be trained without data. As such, data management techniques are also an essential part of computer science education.

Data management is typically taught by means of toy examples [202], with domains such as store ordering systems, movie rentals, and company employees [173]. These examples are accessible: most students have some idea of what data could be involved in these domains, which they can use as a scaffold to remember the database schema. As such, the domain of the data is the inherent context of the database, but determining what makes a good database context has not been studied in-depth before.

We can deepen students' interest in course material by ensuring that the projects they work on are authentic, and by giving them a choice in the what and how of the project [19]. For example, teaching CS1 in the student's major's context has led to increased student success rates [198]. Increased interest in a project may also lead the students to spend more time on an assignment [42]. Furthermore, increased interest makes students become more active learners, which in turn increases motivation and learning [68]. However, it has been shown that more complex databases have the possible downside of making learning SQL more difficult [173].

In this chapter, we examine factors that students think make databases engaging, to answer the question: *What kind of databases do novices deem engaging to study database design, implementation, and querying, and why?* We split this into two research questions:

RQ1 Which factors describe an engaging database application domain for students?

RQ2 Which factors are key to an engaging database in terms of structural complexity?

RQ3 How much exercise data is engaging and why?

To answer these questions, we ran a study in which students designed and implemented their own relational database and queries. We asked them to elaborate on what they thought was engaging about the domain of their database and its complexity. We identified six domain-related factors, five complexity-related factors, and three quantity-related factors. Our findings can help teachers and textbook writers use and create more engaging examples so that students can learn more effectively.

8.2 Related Work

8.2.1 Contextualization and engagement

Whether or not students will benefit from teaching and assignments that are contextualized has been a point of discussion in the computer science education research community for some time now. On one hand, researchers argue that context can increase motivation and reduce student dropout by concretely engaging the student by addressing real-world questions [25, 46, 77, 150]. On the other hand, contextualization can lead to negative side-effects

such as high cognitive load [23] and thinking of newly learned things as only applying in the current situation [69].

In CS education research, contextualization has received attention mainly outside the realm of database education. In perhaps the study most related to this one, Yue used a semi-realistic database (Sakila by MySQL) to increase engagement in their database course by moving away from the typical "toy" example databases [202]. The database was fully integrated into the course and used across multiple assignments. Students found these assignments both interesting and useful, although they also found it difficult because it is more complex than the examples used in textbooks. This increase in database complexity has also been shown to negatively affect successful query formulation among novices [173].

Bouvier et al. ran a study to see whether students perform better on contextualized programming exercises, versus plain, unembellished ones [23]. For this purpose, they made two versions of the same exercise, with and without context. The contextualized one used astronomy as its domain, and the plain one just mentioned numbers. They found that there was no significant difference in performance between the two versions of the exercise [23]. Similar results were also obtained by Leinonen, Denny, and Whalley, who saw that, e.g., students found it easier to implement a linear equation solver when they had context for the problem than without. Furthermore, context led to a longer time on task in seconds, but the contextualized problems also required fewer attempts [98]. Craig, Smith, and Petersen ran a study in which they compared performance on contextualized programming assignments with performance on non-contextualized assignments [47]. In contrast to Leinonen et al., they did not find a measurable effect, leading them to the conclusion that other factors, besides context, impact difficulty and performance more.

Beyond these discussions on performance, it is also interesting to see what students themselves think of contextualization. In 2004, Rich, Perry, and Guzdial ran a study in which they compared a contextualized CS1 course to a standard one [150]. They aimed to attract and retain female students in their CS1 course, and their results suggest that they were successful: female students expressed greater enjoyment and interest in the course than those in the standard course [150]. In a similar vein, Cliburn and Miller found that students liked to have a choice as to how to demonstrate their skill levels, rather than all following the same assignment [42].

It is also worth noting that while contextualization is not a simple concept, measuring its benefits can also be difficult. Rader et al. examined whether students would be more interested in socially relevant contextualized programming tasks than in other types of tasks [145]. In a survey, students responded that they like assignments that show how computer science can be used to benefit society. However, when ranking projects in order of most interesting to least interesting, students showed a higher preference for simpler game-based projects. This preference may be partially explained by the complexity and open-endedness of the humanitarian problem descriptions [145]. Additionally, in the context of computer security education, DeWitt and Cicalese found that students were more engaged with real-world stories than fictitious scenarios. However, context and content need to be in balance to accommodate both traditional computer science students, and those who want to learn to apply techniques in real-world context [53].

Finally, choosing the right context may be difficult. Yarosh and Guzdial applied a narrative contextualization in their media computation CS2 course, in which they used one running example throughout the course [198]. They found that while most of the students agreed that context made the class more interesting, some students found the context taking away from learning the course contents in greater depth. Therefore, engagement is not only about the presence or absence of context. Some other dimensions of engagement include connectedness [58], positive emotions [10, 65, 161], passion [10, 65] and motivation [10, 161].

Engaging assignments can make active students out of passive recipients of knowledge, with active students reaching higher levels of motivation and learning [68]. This leads us to the question of which of these aspects of engagement we can utilize within the context of our database assignments. In this chapter, we focus on student perceptions of engaging database education. With our research questions, we aim to identify factors that educators can use to guide exercise database design, as it remains unclear which factors contribute to engagement in database education.

8.3 Research Setting

8.3.1 Data Collection

The data were collected from three cohorts (springs of 2019-2021) of mainly third-year software engineering and information systems science students at a Finnish university. The students had to have taken a basic course on databases and data management before taking this course. The students were shown a data privacy statement prior to participation, and could then opt-in for the study, where participation carried no incentives or penalties. Out of the 68 students, 56 (82%) chose to participate (15/17 from the first cohort, 8/10 from the second, and 33/41 from the third).

The participants were asked to design and implement a relational database that they deemed engaging for novices who are learning database design and querying. The assignment consisted of multiple tasks, among which the following three are relevant in this chapter:

- Choose an application domain for their database that they deemed engaging for a novice to practice querying, and reflect on why that particular domain is engaging.
- Design the database structure to be engaging for a novice to practice querying, and reflect on why that particular structure is engaging. The minimum requirement for the number of tables is five.
- Fill the database with data and reflect why the amount of data you chose is engaging.

In this study, we analyze the students' reflection reports from the perspective of our research questions. As the language of instruction was Finnish, all questions (including the ones above), databases, and reflections were written in Finnish. All answers were translated through Google Translate and checked by a native speaker.

Ethics. The participants were not financially compensated for participation in the study. The study design and data collection were approved by the Ethical Review Board of the University of Jyväskylä.

8.3.2 **Data analysis**

Of the 56 participants mentioned above, 2 provided the database but did not reflect on their reasoning. These students were not included in the data, so we were left with 54 participants in total. These participants are indicated by P01 through P54.

Our analysis was done in four steps: mining factors, calibration of coding, coding remaining data, and summarizing. We applied directed qualitative content analysis, which is recognized as a viable option when previous literature exists but is deemed incomplete [81].

Mining factors

Before we started our analysis, we used existing literature to create a list of factors to search for in the reflections. We started this approach from the ITiCSE working group report on contextualized programming education [23]. From this report, we retrieved five initial factors: Perceived relevance, Retention, Connection to practice, Impact, and Exploring social and ethical issues. All five of these were used for coding both domain and complexity factors.

Calibration of coding

After setting up this baseline of factors, all authors individually coded the same subset of the data (20%). For this, we used both the existing factors and introduced new factors based on the text in the reflections. Afterward, we aggregated to see where (dis)agreements arose in the coding. To calibrate our coding, we discussed all cases together, as well as any new factors that may have been introduced by any of the authors. New factors were either introduced separately, merged with related factors, or discarded after discussion.

Coding remaining data

Then, two authors split the remaining data in half and coded them individually. For all cases that were not immediately clear, meetings were held to discuss best fits. No new factors were introduced during this time.

Summarizing

After the coding was done, we performed analysis by examining and summarizing all codes per factor. This work was calibrated by working according to an example provided by one of the authors. Within this process of summarizing, we also decided to merge or drop some factors, as they were not contributing to our research question. First, we merged the two types of versatility from both domain and complexity questions, the gist of both was *versatility in difficulty and versatility in extensibility*. We also dropped a complexity factor around the concept of *enough data to practice, no empty result tables*, and a complexity factor where students decided how many columns to create by *intuition*. Neither of the latter two can be used as instructional guidelines, which is why we decided to leave them out.

8.4 **Results**

8.4.1 **Engaging domains**

Database type (# of occurrences)	Database domains (# of occurrences, if more than one)
Support for a physical service (24)	dog contest and health (3); books (2); car sales (2); hotel reservation system (2); university course enrollment (2); bank; board game details and interrelationships; business trip invoicing; car rental service; car wash; employee management; gym memberships; hospital access control; library; multidisciplinary primary school courses; pet health; statistics on board game matches; vaccinations
Delivery of physical or digital goods (21)	online shop (7); digital music platform (5); digital video game distribution platform (4); mobile application store; food delivery platform; marketplace for internet domains; online multiplayer game; operating system update service
Information propagation or collection (14)	statistics on soccer matches (3); dog contest and health (3); academic publications (2); concerts; digital game speedruns; digital mobile gamers; music and movie streaming; pet health; trekking locations
Social interaction (8)	digital video game distribution platform (4); digital music platform (3); car sales

Table 8.1: Database domains and purposes categorized into four themes; note that some domains pertain to more than one category, e.g., there were five databases for digital music platforms, and three of these five databases also contained data structures for social interaction

What were engaging domains?

The participants chose a wide range of business domains for their databases. These are categorized and detailed in Table 8.1. Each database could be categorized into more than one type. The most common type (i.e., deemed most engaging) was a database that supports a physical service, e.g., a database of a bank. The second most popular type was a database that enabled the delivery of physical or digital goods, e.g., a database of an online shop. Third came databases that were intended for information collection or propagation, e.g., a database for academic publications. Finally, some databases were concerned with social interaction, e.g., a database for a vehicle marketplace that provided a forum and chat for buyers and sellers to interact with each other.

In the remainder of this section, we explore the factors we identified as domain-specific in the reflection reports of our students.

D1. Familiarity.

Familiarity, including ease and understandability, was mentioned as a factor by 25 students. They speculated that a database domain that is familiar through, e.g., personal life, previous studies, or work would be engaging. For example, P07 wrote:

> "The database related to listening to music is interesting for the student because almost everyone uses some kind of music streaming service every day."

Several students pointed out that a familiar domain decreases the initial threshold for starting the learning process toward database theory and query writing. P03 wrote:

> "[...] simpler target areas make it easier to internalize new things and maintain interest in the subject more effectively."

On the other hand, a domain not necessarily familiar to many students, but in turn easy to understand, was considered another option for an engaging database domain. In contrast, students highlighted that a domain that would require extensive familiarization would take away from the intended primary learning outcomes of a database course:

> "[...] financial data repository with special terms and complex modeling can cause gray hair for beginners [...] and therefore I don't think is as well suited for training." - P20

D2. Connection to practice.

Related to Familiarity was the factor Connection to practice, mentioned by 30 students. They speculated that understanding the database domain and, e.g., learning to query in this particular domain would also help them in their future work. P07 wrote that:

> "I am interested in the music and entertainment industry in general, so combining the two tells me about my own interests and desire to get a job in the industry."

In contrast, a domain that could be considered niche or rare would be relatively useless outside the scope of a database course. Another consideration included in this factor was the view of increasingly common application domains such as electronic sports and electronic commerce. For example, P01 wrote:

> "[...] useful for working life as it could be real and buying from an online store is growing all the time. This will require more database workers who can handle relational databases in that topic."

D3. Learning opportunity or challenge.

27 students indicated that a domain that provides a learning opportunity or challenge was engaging.

On the one hand, these learning opportunities and challenges could be related to application domains that are complex in terms of the resulting database structure, such that students learn more about database concepts:

> "The student will also learn to design multi-relationship relationships, such as with multiple orders, products, and product categories." - P08

On the other hand, there are learning opportunities to be found in studying the domain itself in order to create correct schemas:

> "Working with a database of publication information can deepen a student's knowledge of scientific publishing, for example, in terms of knowledge of different types of publications." - P06

D4. Versatility

There are two sides to versatility as a factor. The first is versatility in the dimension of query difficulty. This is linked to the concept of a growth mindset for education, and that of "low floor, high ceiling", meaning that assignments are accessible to all students from novices up, but that it also has room to challenge advanced students [20]. The other side is versatility in a database that is easy to expand in size and complexity.

For versatility in query difficulty, ten students argued that this was a factor that made their domain engaging:

> "I wanted to make a position that would be clear enough for new students, as well as that it would contain complex data that could also be used to challenge older students." - P10

> "[...] there are still opportunities for a bit more complex queries, which makes this particularly interesting." - P15

There were eleven students who deemed versatility in terms of a database structure that is easy to expand a factor for engagement. *"I also felt that the topic I chose was broad enough to implement a wide variety of databases, but still easy to summarize."* (P45) With a flexible structure, a novice could mentally select parts of the database, whereas an expert could use all parts.

D5. Personal interest.

The most prominent motivation for rationalizing an engaging domain was personal interest. As many as 33 students argued that the application domain should be interesting for a novice. Students who argued for an interesting domain typically also associated interesting domains with a connection to practice through hobbies, such as P16 and P51:

> "The target area feels pleasant to me, because there will be a lot of gigs and listening to music, so I imagine that it will inspire others in the same way." - P16

> "I chose the simplified online gaming store [...] Examples of a real product are the Valve Steam service or the GOG service. I've been playing video games in my spare time since I was young, so the topic seemed natural." - P51

Other students focused on business ideas they or their friends were setting up, by developing a database for their domain:

> "The target area was the business information of a small company called [anonymized]. This company is really in business, but the business is very small. [...] Against this background, I am building a database structure and possibly a web application program. " - P43

Not unexpectedly, domains chosen because of this justification were diverse in nature.

D6. Social and ethical issues.

Five students deemed that an engaging application domain should address social or ethical issues. Students argued that the application domain should be one that has a seemingly concrete and big impact on our lives.

> "[...] a more serious approach than handling games, for example. If someone makes a small careless mistake in a game character's hat color, no one may even notice it. On the other hand, if a patient has access to hospital drug stores or the surgeon is prevented from entering the operating room, the consequences can be very serious. [...] The database is not designed for entertainment use, but to ensure and promote the safety of people in the hospital environment." - P12

A domain which, due to its important nature, required careful design, was also considered a source of engagement.

D7. Other.

There were five students who argued for something outside of the aforementioned factors. They discussed their chosen domain's engagement with respect to the diversity in data (1), the structure of the data (1), and expanding, complementing, or iterating a software project initiated during another course (3).

> "I wanted to improve my Programming 2 course assignment, Story Statistics, which currently works with text files. That program is still in use by me today and contains a relatively large amount of data, so some version that uses a decent database could be useful at some point, especially if I wanted to develop the structure of the data somehow." - P32

8.4.2 **Engaging complexity**

C1. Simplicity.

A commonly argued characteristic of an engaging database schema, mentioned by 24 participants, was that it should be simple and understandable. P17 summarizes this as:

> "The database had to be interesting for the student so a database that looks too challenging will reduce the student's interest, I think."

The main arguments in favor of simplicity were that it makes it easier to practice writing queries (3 students), to populate the database (1), to expand it (2), and to avoid errors (2), which were also linked to increased complexity due to normalization.

Four students explicitly connected schema simplicity to learning, as *"the complexity of the database could become a barrier to learning"* (P06) and *"in terms of learning, a simple data model is easier to adopt"* (P20). Four students designed a simple database to match their beginner database skills, for example:

> "I wanted to keep the model as simple as possible, as this is the first database I have actually started to implement myself. The main focus was on getting everything to work without a big headache." - P42

C2. Matching information requirements.

36 students described designing their database according to the perceived user or information requirements. Those requirements pertain to the database domain or the software application that would be based on the database. In many cases, students provided a justification per table and relationship and they described the process they followed to come up with the final schema, for example, P30 describes:

> "I started by thinking about the features I would like to have in an imaginary service. I set out to implement the structure of the database through an ER diagram and wondered what items and what kind of relationships the properties I wanted required. I drew an ER diagram and selected the appropriate attributes for the objects and relationships. I transformed the diagram into a relational model according to the transformation rules."

Students commonly described that the database domain led them to the definition of specific attributes for their database, for example:

> "In choosing the number of tables and attributes, I started by thinking about which attributes of the games were essential and important for me to include in the database [..] So I didn't really deliberately choose just that number of tables and columns, but the number of them came as dictated as needed as the plan progressed." - P37

Four students explicitly mentioned matching how a real-world database would be created, for example:

> "In my opinion, the database I have created is useful for working life, because the database is built for a specific real purpose, although the format implemented during the course is not exactly the same as it is or would be in real life." - P16

Some aspects related to information requirements were mentioned rarely as driving the database design. These were the customer requirements (mentioned by P05), enabling search conditions (P24), and supporting database expansion (two students), for example:

> "[...] thought that if publishers or developers ever wanted to expand with new columns, the changes would be easy when they already have separate tables for them." - P27

C3. Learning opportunity or challenge.

Nine students argued that the database schema should support learning. Most of them referred to learning about databases in general or in the context of the databases course, but three of them explicitly referred to learning as preparation for working life; for example, P06 mentioned:

> "[...] in my experience, a database with five tables that meets the minimum requirements for the assignment would be too small to give the student a correct understanding of the databases that will come up in later working life."

Students also gave us insight into how a database schema and its complexity affect learning. A complex database schema enables practicing with *"memorizing conceptual entities"* (P06), encourages the use of a database schema description (the example of the Entity-Relationship diagram was mentioned by a student) when formulating queries, and could

> "[...] demonstrate all the different concepts used in databases (e.g. different types of fact tables in the Kimball model, different types of dimensions in the tables, more metadata columns and tables corresponding to the correct implementation, tables on log data)" - P20

P21 agreed with the learning benefits of a simple database schema:

> "I think learning the basics is supported by moving from simpler schemes to more complex ones. With a simple schema database, it is good to practice making queries and learn to understand logic related to joins, for example."

A complex schema can also make query formulation more challenging, as P55 expresses:

> "Sure, there could have been more tables and columns for the customer's address, for example, but I feel it wouldn't have contributed more to my learning. I think there were enough tables and columns to learn how to use the environment, but not too many to make it too difficult in solving problems."

C4. Versatility.

Eight students mentioned that a database schema should be complicated enough to represent various database characteristics (three students) and to enable practicing with query writing (five students). The database characteristics that students referred to were data types, table relationships, and number of columns, for example:

> "[...] there must be enough columns in at least a few tables to ask meaningful questions. In addition, for queries that use aggregate functions, grouping, and sorting at the same time, it is a good idea to have more columns in the table than the number of functions or expressions used, so that the function, grouping, or sorting can be assigned to different columns and more data-rich" - P06

The database schema should also be versatile enough to support extension for more complex query practice, as summarized by P21:

> "In its current form, the structure of the database is, in my opinion, suitably simple, but at the same time it makes it possible to carry out a wide range of database queries."

C5. Practical constraints.

15 students expressed that in their database design, they took into account practical constraints, specifically assignment restrictions (9), time limitations (4), or challenges with generating model data (2). Assignment restrictions pertain mainly to the minimum number of five tables that was set. The effect of time limitations is described by P51:

> 'The biggest factor limiting the size of the exercise at the moment is time. The work and other courses take up most of my time and I might have liked to build the database more thoroughly."

Finally, P47 mentioned the effect of limitations of the tool that was used for populating the database with data:

> "Mock data contributed in part to the fact that it largely determined the types of data I used in the tables [...] I also added some of the columns because of the effect of the mock data, because as I looked at it I noticed more things I had previously forgotten about in the design."

C6. Other.

Three students mentioned desirable characteristics of the database schema that did not fit into any of the aforementioned ones. Those were to limit the schema to the most significant information and *"focus only on the most important [tables]"* (P08), to match the scope of the course and therefore avoid adding password data *"because with passwords you should be aware of security issues that the course doesn't focus on anyway"* (P27), and to avoid adding calculated columns (P33).

8.4.3 **Engaging amount of data**

The analysis of the student reflections revealed diverse reasons for choosing specific amounts of data. Some participants hand-crafted their data, while others utilized data generators.

A1. Realistic data

The most frequent reason for the chosen amount of data, applied by 31 participants, was the relationship between the database and reality. The main theme mentioned by participants was that the data quantities and ratios should match what they expect from the real world, for example,

> "I tried to keep the ratios of the amount of data in the different tables the same as they would in real life. There are more songs than albums, there are fewer record companies than artists, there are more songwriters than songs, and so on" - Participant 07

However, realistic data should not necessarily be considered a synonym for large amount of data, as i.a. participant 29 chose speedrunning (i.e., competing how fast one can finish a videogame) as the domain, and wrote

> "There are probably quite a small number of registered users, speedrunning is ultimately a pretty niche subculture"

Some participants also deemed that, to engage novices in writing effective queries, there should be enough data for the differences between well or poorly-performing queries to be evident, for example

> "Admittedly, significantly more data would have to be generated if, for example, the performance of different query structures on fact sheets with millions or even billions of rows typical of data warehouses" - Participant 20

So, data should be connected to practice. Which amount is realistic depends on the business domain that the participant chose.

A2. Enough data to practice

Almost as frequently (29 participants) mentioned was the need for enough data to practice database-related topics such as querying. Many participants considered that if correct and incorrect queries return similar result tables, the database does not support engagement. Participant 23, for example, explained that

> "The data had to be such that when an SQL query was written with a logical error, the result table looked different than based on the correct SQL query."

Closely related to the argument of different result tables for different queries, many participants emphasized the importance of creating heterogeneous data, e.g., a dozen customers all from different countries, rather than a hundred from the same country. Participant 20 clarified that

> "The quality, heterogeneity, and representativeness of the data are more important here than the amount of data."

while participant 19 stated that

> "I added data to the database in moderation, but still enough so that the contents, functionality, and special cases of the database could be demonstrated. [...] only a few rows per table could have given the impression that many special cases could not occur with such a small amount of data."

Overall, students feel that the database should have a sufficient amount of data for querying to be meaningful. If an incorrect query produces the same result as a correct query, the data does not facilitate learning, and empty result tables are not engaging. Some special cases of SQL logic, e.g. using expressions on groups, cannot occur with limited amounts of data.

A3. Understandable data

Many participants noted that an engaging exercise database should have data that is conceivable for a database novice. Having a smaller dataset allows for easier inspection of the data, on both logical errors and accuracy:

> "a small amount of data may make it easier to check the accuracy of the query results" - Participant 31

Closely related to the notion that enough data to practice querying is engaging, some participants deemed that smaller datasets make it possible for novices to manually check if their query returns the correct result table, as explained by Participant 52:

> "a smaller amount of data makes a more manageable whole, from which logical errors are revealed more easily because the data can be manually checked".

Participant 1 also added a small amount of data

> "because it keeps the database manageable for learning, and the occurrence of errors is easier to detect. For example, if there were dozens of customers, the result table would be more difficult to read and notice if one person was missing."

In addition to the three reasons discussed above, nine participants also expressed that their reasoning behind designing an engaging amount of data was based simply on *intuition*. Two of these nine participants were not able to give other arguments for their choices.

8.5 Discussion

In subsection 8.3.2 we introduced that our qualitative content analysis was led by both the data and the work by Bouvier et al. on context in CS Education [23]. We will now reflect on whether the factors we identified are in agreement with theirs.

Perceiving relevance. This theme is reflected in multiple identified factors, but most prominently in D3: Learning opportunity or challenge. The students who mentioned this factor were looking to learn about either their domain or databases in more detail, showing that they saw the relevance of these things. We can also argue that it links to D2: Connection to practice and D6: Social and ethical issues, but as we adapted these directly from alternative factors by Bouvier et al. [23], we leave them out of the discussion here.

Retention. This factor did not come up in any of our students' answers. However, this is likely to be an artifact of the assignment as a short-term reflection. We'd like to argue that this theme is implicitly reflected in D3: Learning opportunity or challenge, as students focused on learning which hopefully leads to retention.

Connection to practice. We adopted this theme in our qualitative content analysis and found enough evidence that we kept it as a stand-alone factor.

Impact. Bouvier et al. write about this factor as impact on the broader user community. This factor came up regularly and is mostly coded under Personal interest, where students mention using their assignment database for a start-up by themselves or for helping their friends.

Exploring social and ethical issues. This was kept as a separate factor: D6, but it was mentioned only sporadically. In literature, this is described as a factor that may help female students stay engaged in the material. Unfortunately, as we do not have gender identity data of our students, no conclusions can be drawn on this.

A factor that arose from our findings, and is not mentioned by Bouvier et al. is Versatility. It is worth noting that the two notions of versatility (in business domains (D4), and database complexity (C4)) had subtle but distinct differences. By domain versatility, the students argued for engagement through being able to mentally divide the database into self-contained subsets. This allows students to take the subset that fits their experience level and practice with that. On the other hand, a database with structural versatility was seen to facilitate engagement through being easy to expand.

For their choice of domains, approximately half of the participants chose general and well-understood business domains for their databases. The others chose specific and modern domains such as digital music platforms or mobile application stores. Despite the popularity of social media services, only a few participants chose to include some form of social interaction in their database. It may be that social media was not considered an engaging topic for novices, or it may be that the participants found these domains difficult to design and develop. These speculated difficulties may lie with database structures regarding relationship-heavy characteristics such as social media followers, friendships, and messaging, which could be considered more difficult to model than fact-heavy characteristics such as customers, products, and hotel rooms.

On the performance side of the contextualization discussion, there are some concerns with the cognitive load associated with contextualization. However, we want to note that data management and databases cannot be taught fully de-contextualized: we need data on some domain to fill our databases and inform our queries. As such, it seems better to adapt (some of) the factors that our students wish for than to select something randomly or keep doing what we have always done.

Furthermore, our findings expand on the notion of "useful and interesting" exercise databases presented by Yue, who found that students prefer more realistic databases in learning database-related topics, as these databases are perceived to more closely reflect the databases the students are likely to encounter in their future work [202]. However, Taipalus showed that more complex databases might negatively affect how students learn to query [173]. However, Taipalus also pointed out that more complex databases are not necessarily synonymous with more realistic databases, and student engagement might not depend on realism, but rather on perceived realism. If a teacher can use a relatively simple exercise database that is also argued to be a realistic one, this might be the key to supporting both query formulation learning and engagement.

Our findings support Yue's observations, as several students pointed out that learning opportunities and challenges sustain their engagement. However, we also observed that there are many other, sometimes contradicting factors that all relate to student engagement. Arguably, and although it is possible that a single exercise database may not fulfill all the engagement considerations, teachers should consider the factors presented in subsection 8.4.1 when they design exercise database domains and complexities.

Finally, for our analysis, we chose to work with a database assignment and reflection, instead of the more common questionnaire approach. Rader et al. [145] in their study found that what students say they want to do, differs from what they do. The advantage of our approach is that students had to commit to the domain they chose. They could not arbitrarily select something but had to consider the implications of their choice for the development of the database and queries.

8.5.1 **Practical Implications**

As it seems that students prefer simpler databases and easily understood domains, we should use those in teaching, as querying has been shown to be difficult for novices as is [113, 172],

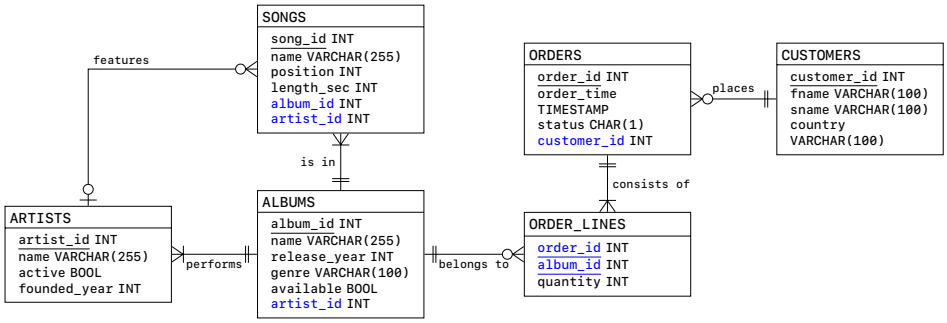


Figure 8.1: An example of an engaging database based on the results yielded by this study and [177]; the domain is relatively easily understood and common, and enables the delivery of digital goods; foreign keys are indicated in blue

without unnecessary complications in database structures, the amount of data, or database domains. On the other hand, there seems to be a school of thought that emphasizes realism and complexity with solid arguments such as preparing students for their future work [190, 202]. These two points of view can also be seen in our data: some participants aimed for engagement through realistic data, while others considered a conceivable or understandable amount more engaging. With these two (often conflicting) viewpoints in mind, many participants argued for *enough* data for querying to be engaging. According to the arguments presented by the participants, enough data is not simply a matter of quantity, but quality as well. That is, there should be enough data to practice querying with meaningful result tables, and the data must be carefully crafted to exhibit heterogeneous values.

On a general level, the database domains in Table 8.1 may serve as guidelines or inspiration for educators who are choosing or designing an exercise database to be used by novices. It may also be worth considering the background of students when deciding whether to create a realistic or understandable amount of data, as some studies have recommended moving from unambiguous tasks to more ambiguous ones as the query writer's skill increases [31, 172].

For a more specific practical implication, based on the results yielded by this study, we constructed a database that should be engaging for novices (Fig. 8.1, structure definitions and data can be downloaded from GitHub[1]). It is worth noting that this database is based on an abstraction and that the domain is selected from a diverse rather than saturated set of domains suggested by the participants. For more information, read our quantitative analysis [177].

For the data, we populated the database presented in Fig. 8.1 with heterogeneous data, so that the novice user may manually check the data to see if their queries contain logical errors. The data were crafted to account for many of the logical query formulation errors described in prior studies [4, 118, 180]. We also designed the database data to contain missing and anomalous values to cater to the demand for realism. This approach has received arguments for [190] and against [180] in prior studies concerned with exercise data.

[1] `https://github.com/tonitaip-2020/engaging_database`

8.5.2 **Limitations and threats to validity**

We note several limitations to our study. First of all, the assignment required all databases to contain at least five tables. Although we do not draw any conclusions from calculated complexity metrics, this requirement was often echoed in students' argumentation on engagement. Given the findings of Rader et al. [145] that students prefer simpler games, and factor C1: Simplicity, we believe that students' behavior might have been different without this restriction.

Second, there is the issue of students' self-selection for participation in this study. Although all students in the course had to complete the exercises we studied in this chapter, it may be that participating students were more motivated to deliver a good result.

Finally, we reflect on the validity of our study based on the definitions by Maxwell [106]. As our study consists of documents containing student argumentation, there is little risk of misunderstanding or misinterpreting students' accounts on the side of the authors (high descriptive validity). However, there is a threat here, as we cannot guarantee that all students understood the questions correctly. One clear example of this issue is shown in factor D5: Personal interest, where some students outlined engagement in terms of their own personal interest, instead of of interest to all novices.

All authors worked together on the qualitative content analysis, leading to a higher likelihood of reliable data coding. Although some interpretation of student answers is required to extract the factors, we believe that the required level of interpretive validity is met, given that our analysis is qualitative and uses quotes from the student answers. Finally, although our sample size can be considered small for quantitative studies, it is on the larger size for qualitative studies. As such, we believe that the factors are representative (but not exhaustive) of other student populations as well.

We want to conclude this discussion section by acknowledging that, although we can gain information about students' leisure pursuits, it is difficult to create appropriate assignments based on this information. To correctly contextualize according to Cooper and Cunningham, we should make the chosen context a theme throughout the course [46], which is a difficult undertaking. One barrier identified by Blumenfeld et al. is that the assignments need to be difficult enough for students to learn about the subject in the required depth [19]. It may be that not all contextualized assignments are of the same difficulty, as Cliburn and Miller hypothesize about the assignments in their work [42]. Rader et al. also concluded that some of their contextualized assignments were too complex and open-ended [145]. Therefore, there is a trade-off to be navigated between how open we can leave assignments to match student interest, and how consistent and in-depth assignments are.

9

Conclusions

In the previous chapters of this dissertation we have presented our new insights to answer the research question introduced in chapter 1: "How can we support students in mastering challenging SQL concepts?" This chapter discusses the main conclusions of our work, their implications for practice, and directions for future research.

9.1 Research Question

The previous chapters of this dissertation presented the results of our investigations in Data Systems Education, answering the research question posed in chapter 1:

How can we support students in mastering challenging SQL concepts?

In chapter 2 we established the complexity of SQL concepts such as (self)-JOINs, correlated subqueries, GROUP BY, and aggregation. Given the large research literature archive on these errors, we can rightfully call them **challenging SQL concepts**. Then, how can we support our students in mastering these? In this dissertation our goal was to understand these difficulties more deeply, and, in the latter half, to develop interventions that could help our students.

In **understanding student struggles**, we identified errors as well as multiple sets of misconceptions. This stream of research led us to develop MSMI1, a multiple-choice questionnaire that can identify misconceptions. This tool can be used by teachers across student populations to identify topics that students have not mastered yet or hold misconceptions about, and through this, support their students.

Then, in **supporting students**, we approached support from another two angles. The first was the development of SQLVis, a visual query representation tool with the goal of reducing students' cognitive load during query formulation. Our quantitative evaluation showed that the students using SQLVis have a higher proportion of correct answers, in fewer attempts and with fewer incorrect answers. As such, it seems that SQLVis supports students in query formulation. Our second project aimed to identify factors influencing engagement in data systems education. This is helpful, because engaged students drop out less, and perform better. The chapter presents changes that teachers and material authors can make to the toy examples they use. Through these changes in their materials, teachers can support their students in performing better on data systems topics such as SQL.

9.2 Summary of contributions

As the title of this dissertation poses, we have attempted to disentangle Data Systems concepts in the context of SQL education. In the section below, we would like to elaborate on three areas of research that we have contributed to: problem-solving, misconceptions, and engagement.

9.2.1 Problem-solving in query formulation

We started this dissertation by exploring problem-solving strategies by our students in chapter 3. Our analysis focused on the complexity of students' solutions. We identified different ways of working in terms of execution order as well as edit distance, but more notably, we uncovered the praxis of mismanaging complexity during formulation: **self-inflicted complexity**. Students hardly start afresh, they have many erroneous attempts, and the complexity of their attempts goes far beyond what is required. This complexity then hinders them in query formulation, as continuous syntax errors lead to error messages that obscure the query's outcome. This in turn may lead to a high cognitive load due to continued translation between natural language and SQL.

One method that can help lower this cognitive load is by using visual representations of the query. The representation can be beneficial, as it has a more explicit visualization of connections that are implicit within SQL itself. Therefore, we designed **SQLVis**, which can generate a graph representation of a subset of queries. In chapter 7, we analyze SQLVis' effectiveness in both a qualitative and a quantitative study. Our quantitative work shows that students are able to work more effectively using SQLVis than without: they needed a lower number of attempts to reach a higher proportion of correct answers.

9.2.2 **Misconceptions**

Another Data Systems Education topic that we have explored in this dissertation is that of misconceptions. Misconceptions are incorrect beliefs about the workings of the language. Although there had been some work on programming misconceptions, none had been mapped for SQL. In chapter 4, we introduced a first study exploring misconceptions for SQL novices. We identified **twelve misconceptions** in four categories.

We then aimed for a broader perspective on the errors and misconceptions to present in chapter 5. SQL experts have a wide range of experiences with writing queries, debugging, pair programming, and error messages. As such, they can provide additional insight into errors. Our expert panel consisted of SQL researchers, teachers, and practitioners. We selected a subset of 11 errors we identified in chapter 4 and presented these to our panel. We asked them to share potential causes for the errors. From their experience, they were able to generate a range of explanations. After sharing these with the whole group, we were left with multiple likely misconceptions that students may suffer from.

To share our findings with the research community, we then developed the **Multiple-choice SQL Misconception Instrument (MSMI1)**, which can be used as a formative assessment for students. In a four-step process, we created and validated the multiple-choice questions we created to identify our students' misconceptions.

Finally, we wanted to measure how representative our findings on misconceptions were. So, we offered the MSMI1 and a set of reflection questions to a diverse student population across universities and continents. We found that all misconceptions in the MSMI are held by a larger student population. In addition, through the reflection questions, we were able to identify an additional **ten misconceptions**.

9.2.3 **Engagement**

We have also contributed to the understanding of student engagement in data systems courses. In chapter 8 we discussed students' descriptions of engaging exercises. We found that database domains that align with students' personal interests are deemed engaging. Concretely, students' chosen database domains most frequently were those supporting a physical service such as hotel reservations or university course enrollment. Concerning complexity, students liked to match the setup of the database to the requirements of the customer. Finally, on the quantity of the data, students had different opinions. The three themes were: realistic data, enough data to practice, and understandable data. This contribution is closed with an **example database** of the appropriate theme and complexity, including a script for setting it up.

9.3 Implications for practice

9.3.1 Problem-solving in query formulation

Our analyses have shown various issues in students' problem-solving behavior. The first is the mismanagement of query complexity. We believe that the underlying cause for this could be a lack of knowledge of SQL syntax and query templates. From the literature, we identified two techniques that could help support students with this in the classroom: retrieval practice and active elaboration [76, p. 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 [199]. However, the application of those techniques to SQL instruction remains to be investigated.

Another cause may be insufficient knowledge of how to decompose the query-building process. 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 [144]. Decomposition can be supported by tracing programs: running the code line-by-line in your head. 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 [91] and SQLVis [112]), as well as concept maps of the evolution process of intermediate results of SQL statements [157]. Adoption of these techniques in the classroom could help students understand query formulation better.

On the problem formulation side, complex problems may be deterring in and of themselves. Lack of practice will result in a lack of skill, so we need to find a way to support students in trying more complex problems. 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 [96] by applying pattern-oriented instruction [86]. In SQL instruction, recent work in this direction has proposed the application of templates to divide-and-conquer query formulation problems [142].

Another theme in our findings was both a lack of practice and a lack of appropriate examples demonstrating SQL concepts, which means students struggled in problem-solving. Such issues can be relieved by increasing the opportunities that students have to interact with SQL, by providing more exercises, as well as more time to learn. Typically, in Computer Science curricula, there are several courses centered around programming. Students are taught about programming languages, design patterns, algorithms, and data structures, and they get the opportunity to practice their programming skills through several courses and projects. Why then do we teach only the basics with regard to Databases? With the increasing need for Data Science topics in Computer Science curricula, there should be more focus on teaching students about data storage and retrieval via a reduction of topics per course, to allow for more opportunities for practice with the topics that are included.

9.3.2 **Misconceptions**

There are also various problems in teaching and learning that may lead to misconceptions. Starting from the student side, the first problem is incorrect knowledge transfer. Transfer happens when students connect information and new concepts to existing knowledge, but misinterpretations of knowledge can cause incorrect connections. We can identify faulty transfer through the uncovering of misconceptions. This is an indication of where course material and instruction need to be adapted to the students' previous knowledge. Teachers who have an understanding of misconceptions can work towards knowledge refinement and reorganization, to build students' knowledge of the taught concepts [163].

Many of the common misconceptions are those that have to do with data relationships and table joins. For these concepts, teachers can help students by offering the use of tools that visualize data relationships or intermediate query results as in [33, 59, 100, 112].

Techniques for preventing or mitigating these misconceptions are under-explored for SQL. One promising technique is utilizing refutation texts. These are texts that point out the misconception and explain why it is incorrect. They consist of either two or three elements: a misconception, an explanation of the correct concept, and (optionally) a cue, which helps the student understand that the misconception is incorrect [185]. Once these have developed, they can be applied in the classroom to reduce misconception formation.

Finally, the black-box view that some students have of the DBMS is a problem, as such a view does not promote understanding. With this perspective, students (consciously or unconsciously) avoid having to develop an intricate mental model of the DBMS: the black box representation does not facilitate mental model construction [89]. This behavior might be rooted in avoidance or anxiety if the student consciously avoids it, but may also reflect a lack of interest or growth mindset in its accidental application. In case of the former, teachers can offer students tools and representation models to make the material more accessible and offer meaningful learning [89]. In case of a lack of interest or growth mindset, teachers should focus more on promoting engagement with the material.

9.3.3 **Engagement**

In this dissertation, we have identified factors that students believe make an engaging database. Teachers should reflect on the toy examples they use in class and evaluate them against these factors. As many teachers are likely to use textbooks to gather the toy examples and exercises they have students practice with, the findings in this thesis are also of interest to data systems textbook authors.

What remains is a trade-off between simple and more realistic database schemas. Students may have a preference for either, as our results in chapter 8 have shown. Using the correct level of complexity to match the student's interests can help the student learn by utilizing Vygotsky's zone of proximal development: what a student cannot do alone yet, but can achieve through guidance, and then later has developed the skills to perform alone [34]. Some studies have recommended moving from unambiguous tasks to more ambiguous ones as the query writer's skill increases [31, 172]. Offering different schemas and levels of exercises is likely to improve the learning process for many students.

Finally, based on the results yielded by this study, we constructed a database that should be engaging for novices (Fig. 8.1, structure definitions and data can be downloaded from GitHub[1]). Teachers can use this database as a shortcut to running the evaluations suggested above.

9.4 Future work

During our work on the contributions in this thesis, we identified many other research directions that drew our attention. Below, we will explain some elements worth exploring.

9.4.1 Decomposition, subgoals, and notional machines

One intrinsic complication in teaching SQL is its declarative nature. Research on programming misconceptions in imperative languages has shown that decomposition of problem descriptions is difficult for students [144]. One way in which a student can decompose a problem is to trace the execution of that problem. For SQL, this is even more difficult as execution in declarative languages cannot be traced, because query plan generation is nondeterministic. This characteristic of SQL may limit the extent to which a student can generate an appropriate mental model of query execution. Thus, studying decomposition and subgoals in query formulation is essential to understanding the (partial) mental models that students construct about SQL.

Specific interventions could be researched for helping novices decompose SQL problems and manage complexity, such as teaching SQL patterns and templates. 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.

In addition, it could be highly useful to research a notional machine for declarative languages. Notional machines "are representations or analogies that put a spotlight on those things that are important to look at, or that do something that makes apparent otherwise invisible behavior which, if un-noticed or misunderstood, would cause the learner to go hopelessly wrong." [57, p. 22]. If we were to develop a notional machine for SQL, this could give us more insight into the link between student misconceptions, student errors, and DBMS processing.

9.4.2 Teaching inventory

Notional machines are not only related to misconceptions but also to what we teach. If the teacher can leverage the appropriate analogies, this may lead to improper mental models. Therefore, another interesting avenue of future work is to identify what data systems lecturers teach and how. A large interview study or focus group, potentially combined with the Delphi protocol, could give us insights into topics of importance and the preference of teaching methodology. We can combine this study with a revisiting of instructional approaches and materials, to understand their (in)adequacy in effectively addressing misconceptions.

[1] https://github.com/tonitaip-2020/engaging_database

In exploring teaching material, we can also study how existing teaching materials fare when compared to our recommendations from chapter 8. Furthermore, it would be interesting to see what the effect of adopting our recommendations is on student engagement, and possibly performance. These results can be utilized by educators and textbook authors in designing exercise databases that cater for novice engagement.

9.4.3 **Database anxiety**

Engagement can be considered the inverse of the concept we describe as database anxiety in chapter 5. Database anxiety is an avoidance behavior similar to math anxiety [9], where highly math-anxious students tend to withdraw from math practice, creating a perpetual cycle that damages any existing math competence. A similar process may occur in SQL querying when students need to apply more complex concepts such as JOINs, aggregation, and subqueries that they may shy away from. The extent to which database anxiety occurs in our student population, and to which extent it influences student performance is a promising topic of future work.

9.4.4 **SQLVis**

Complex concepts should not just be investigated in light of database anxiety. As Taipalus and Seppänen write, advanced SQL concepts are seldom included in educational studies [179]. When we reflect on SQLVis' implementation, the same holds here. Given the limited time and page limit, we focused on a basic set of features at first. However, newer work should focus on expanding SQLVis with additional keywords and logic.

Furthermore, SQLVis does not work in case of syntax errors, which meant that participants using it in our study struggled extra with these errors. Therefore, visual representations of syntax errors to support learners are an important topic for further study.

Lastly, it would be valuable to study the effect of SQLVis usage more deeply. For example, does the representation really reduce cognitive load, or are there other mechanisms at play during query formulation? In such a study, we should also move beyond studying the population at a single university, allowing us to show that SQLVis works in multiple contexts.

9.4.5 **MSMI2**

Finally, in this dissertation, we presented MSMI1, our first version of a SQL Misconception Instrument. It would be valuable to expand this instrument with additional questions to capture even more misconceptions. The development of questions for the remaining identified misconceptions is a first step. We will likely identify more misconceptions during the process of validation (as we did in chapter 6), so this is likely to become an iterative process.

References

[1] **Abouzied, A., Hellerstein, J. M., and Silberschatz, A.** Playful Query Specification with DataPlay. In *Proceedings of the VLDB Endowment* (2012), pp. 1938–1941.

[2] **Ahadi, A., Behbood, V., Vihavainen, A., Prior, J., and Lister, R.** 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)* (2016), pp. 401–406.

[3] **Ahadi, A., Prior, J., Behbood, V., and Lister, R.** 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)* (2015), pp. 201–206.

[4] **Ahadi, A., Prior, J., Behbood, V., and Lister, R.** 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)* (2016), pp. 272–277.

[5] **Aivaloglou, E., Fletcher, G., Liut, M., and Miedema, D.** Report on the First International Workshop on Data Systems Education (DataEd'22). *ACM SIGMOD Record 51*, 4 (2023), 49–53.

[6] **Aivaloglou, E., Fletcher, G., and Miedema, D.** DataEd'22-1st International Workshop on Data Systems Education: Bridging Education Practice with Education Research. In *Proceedings of the 2022 International Conference on Management of Data* (2022), pp. 2556–2557.

[7] **Aivaloglou, E., Fletcher, G., and Miedema, D.** DataEd'23 - 2nd International Workshop on Data Systems Education: Bridging Education Practice with Education Research. In *SIGMOD '23: Companion of the 2023 International Conference on Management of Data* (Seattle, USA, 2023), pp. 313–314.

[8] **Arakawa, K., Hao, Q., Greer, T., Ding, L., Hundhausen, C. D., and Peterson, A.** In situ identification of student self-regulated learning struggles in programming assignments. *SIGCSE 2021 - Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (2021), 467–473.

[9] **Ashcraft, M. H.** Math anxiety: Personal, educational, and cognitive consequences. *Current directions in psychological science 11*, 5 (2002), 181–185.

[10] **Barkley, E. F., and Major, C. H.** *Student engagement techniques: A handbook for college faculty*. John Wiley & Sons, 2020.

[11] **Bateson, A. G., Alexander, R. A., and Murphy, M. D.** Cognitive processing differences between novice and expert computer programmers. *International Journal of Man-Machine Studies 26*, 6 (1987), 649–660.

[12] **Bayman, P., and Mayer, R. E.** A diagnosis of beginning programmers' misconceptions of basic programming statements. *Communications of the ACM 26*, 9 (Sept. 1983), 677–679.

[13] **Beck, F., Burch, M., and Diehl, S.** Towards an aesthetic dimensions framework for dynamic graph visualisations. *Proceedings of the International Conference on Information Visualisation* (2009), 592–597.

[14] **Beker, K., Kim, J., Van Boekel, M., van den Broek, P., and Kendeou, P.** Refutation texts enhance spontaneous transfer of knowledge. *Contemporary Educational Psychology 56* (2019), 67–78.

[15] **Bennett, C., Ryall, J., Spalteholz, L., and Gooch, A.** The aesthetics of graph visualization. *Computational Aesthetics in Graphics, Visualization, and Imaging* (2007), 1–8.

[16] **Bhowmick, S. S.** DB x HCI: towards bridging the chasm between graph data management and HCI. In *Database and Expert Systems Applications* (2014), Springer, pp. 1–11.

[17] **Biggs, J.** Enhancing teaching through constructive alignment. *Higher education 32*, 3 (1996), 347–364.

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

[19] **Blumenfeld, P. C., Soloway, E., Marx, R. W., Krajcik, J. S., Guzdial, M., and Palincsar, A.** Motivating project-based learning: Sustaining the doing, supporting the learning. *Educational Psychologist 26*, 3-4 (1991), 369–398.

[20] **Boaler, J., Dieckmann, J. A., LaMar, T., Leshin, M., Selbach-Allen, M., and Pérez-Núñez, G.** The transformative impact of a mathematical mindset experience taught at scale. *Frontiers in Education 6* (2021), 1–13.

[21] **Boley, H.** Grailog 1.0: Graph-Logic Visualization of Ontologies and Rules. In *International Workshop on Rules and Rule Markup Languages for the Semantic Web.* (2013).

[22] **Boulay, B. D.** Some difficulties of learning to program. *Journal of Educational Computing Research 2*, 1 (1986), 57–73.

[23] **Bouvier, D., Lovellette, E., Matta, J., Alshaigy, B., Becker, B. A., Craig, M., Jackova, J., McCartney, R., Sanders, K., and Zarb, M.** Novice programmers and the problem description effect. In *Proceedings of the 2016 ITiCSE Working Group Reports* (New York, NY, USA, 2016), ITiCSE '16, Association for Computing Machinery, p. 103–118.

[24] **Bowen, P. L., Ferguson, C. B., Lehmann, T. H., and Rohde, F. H.** Cognitive style factors affecting database query performance. *International Journal of Accounting Information Systems 4* (2003), 251–273.

[25] **Brannock, E., Lutz, R., and Napier, N.** Integrating authentic learning into a software development course: An experience report. In *Proceedings of the 14th annual ACM SIGITE conference on Information technology education* (2013), pp. 195–200.

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

[27] **Brown, N. C. C., and Altadmri, A.** Novice java programming mistakes: Large-scale data vs. educator beliefs. *ACM Transactions on Computing Education 17*, 2 (May 2017).

[28] **Buitendijk, R. B.** Logical Errors in Database SQL Retrieval Queries. *Computer Science in Economics and Management 1* (1988), 79–96.

[29] **Caceffo, R., Wolfman, S., Booth, K. S., and Azevedo, R.** Developing a computer science concept inventory for introductory programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (2016), pp. 364–369.

[30] **Cass, S.** Top programming languages 2022. `https://spectrum.ieee.org/top-programming-languages-2022`, 2022. [Online; accessed 24 August 2023].

[31] **Casterella, G. I., and Vijayasarathy, L.** An Experimental Investigation of Complexity in Database Query Formulation Tasks. *Journal of Information Systems Education 24*, 3 (2013), 211–221.

[32] **Catarci, T., Mecella, M., Kimani, S., and Santucci, G.** Visual Query Interfaces. In *The Wiley Handbook of Human Computer Interaction*. John Wiley & Sons, Ltd, Chichester, UK, dec 2017, pp. 561–577.

[33] **Cembalo, M., De Santis, A., and Umberto, F. P.** SAVI: A new system for advanced SQL visualization. In *Proceedings of the 2011 ACM Special Interest Group for Information Technology Education Conference* (New York, NY, USA, 2011), Association for Computing Machinery, pp. 165–170.

[34] **Chaiklin, S.** The zone of proximal development in Vygotsky's analysis of learning and instruction. In *Vygotsky's educational theory in cultural context*, A. Kozulin, B. Gindis, V. Ageyev, and S. Miller, Eds. Cambridge, UK: Cambridge University Press., 2003, pp. 39–64.

[35] **Chamberlin, D. D.** Early history of SQL. *IEEE Annals of the History of Computing 34*, 4 (2012), 78–82.

[36] **Chamberlin, D. D., and Boyce, R. F.** Sequel: A structured english query language. In *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control* (1974), pp. 249–264.

[37] **Chan, H. C.** A Two-stage Evaluation of User Query Performance for the Relational Model and SQL. In *PACIS 2007 Proceedings* (2007), pp. 213–227.

[38] **Chan, H. C., Tan, B. C. Y., and Wei, K. K.** Three important determinants of user performance for database retrieval. *International Journal of Human-Computer Studies 51*, 5 (nov 1999), 895–918.

[39] **Chein, M., and Mugnier, M.-L.** Conceptual Graphs with Negation. In *Graph-based Knowledge Representation*, L. Jain and X. Wu, Eds. Springer-Verlag London, 2009, ch. 12, pp. 337–377.

[40] **Clancy, M.** Misconceptions and attitudes that interfere with learning to program. In *Computer Science Education Research*. Taylor & Francis Group, 2004, ch. 1, pp. 85–100.

[41] **Clayton, M. J.** Delphi: A technique to harness expert opinion for critical decision-making tasks in education. *Educational Psychology 17*, 4 (1997), 373–386.

[42] **Cliburn, D. C., and Miller, S.** Games, stories, or something more traditional: The types of assignments college students prefer. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 2008), SIGCSE '08, Association for Computing Machinery, p. 138–142.

[43] **Codd, E. F.** A relational model of data for large shared data banks. *Communications of the ACM 13*, 6 (1970), 377 – 387.

[44] **Collaris, D., Weerts, H. J., Miedema, D., van Wijk, J. J., and Pechenizkiy, M.** Characterizing Data Scientists' Mental Models of Local Feature Importance. In *Nordic Human-Computer Interaction Conference* (Aarhus, Denmark, 2022), pp. 1–12.

[45] **Cook, M. P.** Visual Representations in Science Education: The Influence of Prior Knowledge and Cognitive Load Theory on Instructional Design Principles. *Science Education 90*, 6 (2007), 1073–1091.

[46] **Cooper, S., and Cunningham, S.** Teaching computer science in context. *ACM Inroads 1*, 1 (2010), 5–8.

[47] **Craig, M., Smith, J., and Petersen, A.** Familiar contexts and the difficulty of programming problems. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research* (New York, NY, USA, 2017), Koli Calling '17, Association for Computing Machinery, p. 123–127.

[48] **Dadashzadeh, M.** A Simpler Approach to Set Comparison Queries in SQL. *Journal of Information Systems Education 14*, 4 (2003), 345–348.

[49] **Dasgupta, S., Hale, W., Monroy-Hernández, A., and Hill, B. M.** Remixing as a pathway to computational thinking. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (2016), pp. 1438–1449.

[50] **De Raadt, M., Dekeyser, S., and Lee, T. Y.** Do students SQLify? Improving learning outcomes with peer review and enhanced computer assisted assessment of querying skills. *ACM International Conference Proceeding Series 276* (2006), 101–108.

[51] **Denny, P., and Fischer, K.** What's in scope at ICER. `https://icer2023.acm.org/track/icer-2023-papers#Author-Guidelines`, 2023. [Online; accessed 24 August 2023].

[52] **Denny, P., Luxton-Reilly, A., and Simon, B.** Evaluating a new exam question: Parsons problems. In *Proceedings of the fourth international workshop on computing education research* (2008), pp. 113–124.

[53] **DeWitt, J., and Cicalese, C.** Contextual integration: A framework for presenting social, legal, and ethical content across the computer security and information assurance curriculum. In *Proceedings of the 3rd annual conference on Information security curriculum development* (2006), pp. 30–40.

[54] **Duran, R., Sorva, J., and Leite, S.** Towards an analysis of program complexity from a cognitive perspective. In *Proceedings of the 14th ACM Conference on International Computing Education Research (ICER)* (2018), p. 21–30.

[55] **Eccles, D. W., and Arsal, G.** The think aloud method: what is it and how do i use it? *Qualitative Research in Sport, Exercise and Health 9*, 4 (2017), 514–531.

[56] **Farghally, M. F., Koh, K. H., Ernst, J. V., and Shaffer, C. A.** Towards a concept inventory for algorithm analysis topics. In *Annual Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2017), Association for Computing Machinery, pp. 207–212.

[57] **Fincher, S., Jeuring, J., Miller, C. S., Donaldson, P., du Boulay, B., Hauswirth, M., Hellas, A., Hermans, F., Lewis, C., Mühling, A., Pearce, J. L., and Petersen, A.** Notional machines in computing education: The education of attention. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2020), ITiCSE-WGR '20, Association for Computing Machinery, p. 21–50.

[58] **Fletcher, A.** Defining student engagement: A literature review. `https://soundout.org/2015/03/29/defining-student-engagement-a-literature-review/`, 2015. [Online; Accessed 25 Aug 2023].

[59] **Folland, K. A. T.** *viSQLizer: An interactive visualizer for learning SQL*. PhD thesis, Norwegian University of Science and Technology, 2016.

[60] **Gathani, S., Lim, P., and Battle, L.** Debugging database queries: A survey of tools, techniques, and users. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (2020), p. 1–16.

[61] **Gatterbauer, W.** Databases will Visualize Queries too. *Proceedings of the VLDB Endowment 4*, 12 (2011).

[62] **Gibson, H., Faith, J., and Vickers, P.** A survey of two-dimensional graph layout techniques for information visualisation. *Information Visualization 12*, 3-4 (2013), 324–357.

[63] **Goldman, K., Gross, P., Heeren, C., Herman, G., Kaczmarczyk, L., Loui, M. C., and Zilles, C.** Identifying important and difficult concepts in introductory computing courses using a Delphi process. In *Proceedings of the 39th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2008), Association for Computing Machinery, pp. 256–260.

[64] **Gray, G. L., Evans, D., Cornwell, P., Costanzo, F., and Self, B.** Toward a nationwide dynamics concept inventory assessment test. In *ASEE Annual Conference Proceedings* (2003), pp. 9187–9198.

[65] **Great Schools Partnership**. Student engagement. `https://www.edglossary.org/student-engagement/`, 2016.

[66] **Greene, S. L., Devlin, S. J., Cannata, P. E., and Gomez, L. M.** No ifs, ands, or ors: A study of database querying. *International Journal of Man-Machine Studies 32*, 3 (Mar. 1990), 303–326.

[67] **Grillenberger, A., and Brinda, T.** eledSQL - A New Web-Based Learning Environment for Teaching Databases and SQL at Secondary School Level. In *Proceedings of the 7th Workshop in Primary and Secondary Computing Education* (2012), p. 101–104.

[68] **Groccia, J. E.** What is student engagement? *New directions for teaching and learning 2018*, 154 (2018), 11–20.

[69] **Guzdial, M.** Does Contextualized Computing Education Help? *ACM Inroads 1*, 4 (Dec. 2010), 4–6.

[70] **Hardt, R., and Gutzmer, E.** Database query analyzer (DBQA) - A data-oriented SQL clause visualization tool. *SIGITE 2017 - Proceedings of the 18th Annual Conference on Information Technology Education* (2017), 147–152.

[71] **Hasan, S., Bagayoko, D., and Kelley, E. L.** Misconceptions and the certainty of response index (CRI). *Physics Education 34*, 5 (1999), 294–299.

[72] **Helminen, J., Ihantola, P., Karavirta, V., and Malmi, L.** How do students solve parsons programming problems? an analysis of interaction traces. In *Proceedings of the ninth annual international conference on International computing education research* (2012), pp. 119–126.

[73] **Herman, G. L.** *The Development of a Digital Logic Concept Inventory*. PhD thesis, University of Illinois at Urbana-Champaign, 2011.

[74] **Hermans, F.** Hedy: A Gradual Language for Programming Education. *ICER 2020 - Proceedings of the 2020 ACM Conference on International Computing Education Research* (2020), 259–270.

[75] **Hermans, F.** Decoding your confusion while coding. In *The Programmer's Brain*. Manning Publications, 2021, ch. 1.

[76] **Hermans, F.** How to learn programming syntax quickly. In *The Programmer's Brain*. Manning Publications, 2021, ch. 3.

[77] **Herrmann, N., and Popyack, J. L.** Creating an authentic learning experience in introductory programming courses. In *Proceedings of the twenty-sixth SIGCSE technical symposium on Computer Science Education* (1995), pp. 199–203.

[78] **Hestenes, D., Wells, M., and Swackhamer, G.** Force concept inventory. *The physics teacher 30*, 3 (1992), 141–158.

[79] **Hill, H. C., Ball, D. L., and Schilling, S. G.** Unpacking pedagogical content knowledge: Conceptualizing and measuring teachers' topic-specific knowledge of students. *Journal for Research in Mathematics Education JRME 39*, 4 (2008), 372 – 400.

[80] **Hristova, M., Misra, A., Rutter, M., and Mercuri, R.** Identifying and Correcting Java Programming Errors for Introductory Computer Science Students. *SIGCSE Bull. 35*, 1 (Jan. 2003), 153–156.

[81] **Hsieh, H.-F., and Shannon, S. E.** Three approaches to qualitative content analysis. *Qualitative Health Research 15*, 9 (2005), 1277–1288.

[82] **Huang, W., Eades, P., and Hong, S. H.** Measuring effectiveness of graph visualizations: A cognitive load perspective. *Information Visualization 8*, 3 (2009), 139–152.

[83] **Huang, W., Hong, S., and Eades, P.** Predicting graph reading performance: a cognitive approach. In *Asia-Pacific Symposium on Information Visualisation* (2006), pp. 207–216.

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

[85] **ISO-9075:1987**. Information processing systems — Database language — SQL. Standard, International Organization for Standardization, 1987. `https://www.iso.org/standard/16661.html`.

[86] **Iyer, V., and Zilles, C.** Pattern Census: A Characterization of Pattern Usage in Early Programming Courses. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE)* (2021), p. 45–51.

[87] **Jagadish, H., Chapman, A., Elkiss, A., Jayapandian, M., Li, Y., Nandi, A., and Yu, C.** Making database systems usable. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data* (2007), pp. 13–24.

[88] **Jin, C., Bhowmick, S. S., Choi, B., and Zhou, S.** Prague: towards blending practical visual subgraph query formulation and query processing. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on* (2012), IEEE, pp. 222–233.

[89] **Jonassen, D. H., and Strobel, J.** Modeling for meaningful learning. In *Engaged learning with emerging technologies*, D. Hung and M. S. Khine, Eds. Springer, 2006, ch. 1.

[90] **Kaczmarczyk, L. C., Petrick, E. R., East, J. P., and Herman, G. L.** Identifying student misconceptions of programming. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2010), SIGCSE '10, Association for Computing Machinery, p. 107–111.

[91] **Kearns, R., Shead, S., and Fekete, A.** A teaching system for SQL. In *Proceedings of the 2nd Australasian conference on Computer Science Education (ACSE)* (1997), pp. 224–231.

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

[93] **Kindlmann, G., and Scheidegger, C.** An Algebraic Process for Visualization Design. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (dec 2014), 2181–2190.

[94] **Kleerekoper, A., and Schofield, A.** SQL tester: an online SQL assessment tool and its impact. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE 2018* (New York, New York, USA, 2018), ACM Press, p. 87–92.

[95] **Ko, A. J., and Myers, B. A.** Development and evaluation of a model of programming errors. In *IEEE Symposium on Human Centric Computing Languages and Environments, 2003. Proceedings. 2003* (2003), pp. 7–14.

[96] **Kranch, D.** Teaching the novice programmer: A study of instructional sequences and perception. *Education and Information Technologies 17* (09 2012), 291–313.

[97] **Kuechler, W., and Simkin, M.** How well do multiple choice tests evaluate student understanding in computer programming classes? *Journal of Information Systems Education 14*, 4 (2003), 389.

[98] **Leinonen, J., Denny, P., and Whalley, J.** Exploring the effects of contextualized problem descriptions on problem solving. In *Australasian Computing Education Conference* (New York, NY, USA, 2021), ACE '21, Association for Computing Machinery, p. 30–39.

[99] **Levenshtein, V. I., et al.** Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady* (1966), vol. 10, Soviet Union, pp. 707–710.

[100] **Leventidis, A., Zhang, J., Dunne, C., Gatterbauer, W., Jagadish, H., and Riedewald, M.** QueryVis: Logic-based diagrams help users understand complicated SQL queries faster. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2020), Association for Computing Machinery, pp. 2303–2318.

[101] **Ma, L.** *Investigating and Improving Novice Programmers' Mental Models of Programming Concepts*. PhD thesis, University of Strathclyde, 2007.

[102] **Ma, L., Ferguson, J., Roper, M., and Wood, M.** Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education 21*, 1 (2011), 57–80.

[103] **Margulieux, L., Denny, P., Cunningham, K., Deutsch, M., and Shapiro, B. R.** When Wrong is Right : The Instructional Power of Multiple Conceptions. In *Proceedings of the 17th ACM Conference on International Computing Education Research* (New York, NY, USA, 2021), Association for Computing Machinery, pp. 184–197.

[104] **Mathon, L., and Miedema, D.** Increasing Awareness of SQL Anti-Patterns for Novices: A Study Design. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 2* (Lugano, Switzerland, 2022), pp. 42–43.

[105] **Matos, V. M., and Grasser, R.** A Simpler (and Better) SQL Approach to Relational Division. *Journal of Information Systems Education 13*, 2 (2002), 85–88.

[106] **Maxwell, J. A.** Understanding and Validity in Qualitative Reseach. *Harvard Educational Review 62*, 3 (1992), 279–300.

[107] **Mayer, R. E., and Moreno, R.** Nine Ways to Reduce Cognitive Load in Multimedia Learning. *Educational Psychologist 38*, 1 (2010), 43–52.

[108] **Miedema, D.** Towards successful interaction between Humans and Databases. Master's thesis, Eindhoven University of Technology, 2019.

[109] **Miedema, D.** Toward a Fundamental Understanding of SQL Education. In *Proceedings of the 2023 ACM Conference on International Computing Education Research V.2* (Chicago, USA, 2023).

[110] **Miedema, D., Aivaloglou, E., and Fletcher, G.** Exploring the prevalence of SQL misconceptions: a study design. In *Proceedings of the 21st Koli Calling International Conference on Computing Education Research* (Virtual event, 2021), pp. 1–3.

[111] **Miedema, D., Aivaloglou, E., and Fletcher, G.** Identifying SQL Misconceptions of Novices: Findings from a Think-Aloud Study. In *Proceedings of the 17th ACM Conference on International Computing Education Research* (Virtual Event, USA, 2021), pp. 355–367. This publication was reprinted in ACM Inroads as "Worth reading from ICER 2021".

[112] **Miedema, D., and Fletcher, G.** SQLVis: Visual Query Representations for Supporting SQL Learners. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (Virtual Event, USA, 2021), IEEE Computer Society, pp. 1–9.

[113] **Miedema, D., Fletcher, G., and Aivaloglou, E.** Expert Perspectives on Student Errors in SQL. *ACM Transactions on Computing Education (TOCE) 23*, 1 (2022), 1–28.

[114] **Miedema, D., Fletcher, G., and Aivaloglou, E.** So many brackets! An analysis of how SQL learners (mis)manage complexity during query formulation. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension* (Virtual event, 2022), pp. 122–132.

[115] **Miedema, D., Liut, M., Fletcher, G., and Aivaloglou, E.** MSMI1: Towards a Validated SQL Misconceptions Instrument. In *Proceedings of the 2023 ACM Conference on International Computing Education Research V.2* (Chicago, USA, 2023).

[116] **Miedema, D., Liut, M., Fletcher, G., and Aivaloglou, E.** "There is no ambiguity on what to return": Investigating the Prevalence of SQL Misconceptions. In *23rd Koli Calling International Conference on Computing Education Research* (Koli, Finland, 2023).

[117] **Miedema, D., Taipalus, T., and Aivaloglou, E.** Students' Perceptions on Engaging Database Domains and Structures. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto, Canada, 2023), pp. 122–128.

[118] **Migler, A., and Dekhtyar, A.** Mapping the SQL learning process in introductory database courses. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE)* (2020), pp. 619–625.

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

[120] **Moreno-León, J., Robles, G., and Román-González, M.** Comparing computational thinking development assessment scores with software complexity metrics. In *2016 IEEE Global Engineering Education Conference (EDUCON)* (2016), IEEE, pp. 1040–1045.

[121] **Morrison, B. B., Margulieux, L. E., Ericson, B., and Guzdial, M.** Subgoals help students solve Parsons problems. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (2016), pp. 42–47.

[122] **Muñoz Barón, M., Wyrich, M., and Wagner, S.** An empirical validation of cognitive complexity as a measure of source code understandability. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (2020), pp. 1–12.

[123] **Nagataki, H., Nakano, Y., Nobe, M., Tohyama, T., and Kanemune, S.** A visual learning tool for database operation. *ACM International Conference Proceeding Series* (2013), 39–40.

[124] **Nagy, C., and Cleve, A.** A Static Code Smell Detector for SQL Queries Embedded in Java Code. *Proceedings - 2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation, SCAM 2017 2017-Octob* (2017), 147–152.

[125] **Nelson, G. L., Xie, B., and Ko, A. J.** Comprehension first: Evaluating a novel pedagogy and tutoring system for program tracing in CS1. *ICER 2017 - Proceedings of the 2017 ACM Conference on International Computing Education Research* (2017), 2–11.

[126] **Obaido, G., Ade-Ibijola, A., and Vadapalli, H.** Generating SQL queries from visual specifications. *Communications in Computer and Information Science 963* (2019), 315–330.

[127] **Obionwu, V., Broneske, D., Hawlitschek, A., Köppen, V., and Saake, G.** SQLValidator – An Online Student Playground to Learn SQL. *Datenbank-Spektrum 21*, 2 (2021), 73–81.

[128] **Odynchuk, A., Miedema, D., and Fletcher, G.** SQLVis+: Restoring Visual Query Representations in Case of Syntax Errors. Under review.

[129] **Ogden, W. C., and Brooks, S. R.** Query languages for the casual user. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI)* (New York, New York, USA, 1983), ACM Press, pp. 161–165.

[130] **Orlov, D., Miedema, D., and Yakovets, N.** A Cup of ChaiSQL: Benefits of Type Checking for SQL. Under review.

[131] **Ormerod, T.** Human cognition and programming. In *Psychology of programming*. Elsevier, 1990, pp. 63–82.

[132] **Park, S., and Oliver, J. S.** Revisiting the conceptualisation of pedagogical content knowledge (pck): Pck as a conceptual tool to understand teachers as professionals. *Research in Science Education 38*, 3 (2008), 261–284.

[133] **Parker, M. C., Guzdial, M., and Engleman, S.** Replication, validation, and use of a language independent cs1 knowledge assessment. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (Melbourne VIC Australia, Aug 2016), ACM, p. 93–101.

[134] **Paul, W., and Vahrenhold, J.** Hunting high and low: instruments to detect misconceptions related to algorithms and data structures. In *Proceeding of the 44th ACM technical symposium on Computer science education* (Denver Colorado USA, Mar 2013), ACM, p. 29–34.

[135] **Pea, R. D.** Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research 2*, 1 (1986), 25–36.

[136] **Piattini, M., and Martinez, A.** Measuring for database programs maintainability. In *Proceedings of the 11th International Conference on Database and Expert Systems Applications (DEXA)* (Berlin, Heidelberg, 2000), M. Ibrahim, J. Küng, and N. Revell, Eds., Springer Berlin Heidelberg, pp. 65–78.

[137] **Porter, L., Zingaro, D., Liao, S. N., Taylor, C., Webb, K. C., Lee, C., and Clancy, M.** BDSI: A Validated Concept Inventory for Basic Data Structures. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON Canada, Jul 2019), ACM, p. 111–119.

[138] **Poulsen, S., Butler, L., Alawini, A., and Herman, G. L.** Insights from Student Solutions to SQL Homework Problems. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE* (2020), 404–410.

[139] **Presler-Marshall, K., Heckman, S., and Stolee, K. T.** 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)* (2021), IEEE, pp. 199–210.

[140] **Purchase, H. C., Hoggan, E., and Görg, C.** How important is the "mental map"? - An empirical investigation of a dynamic graph layout algorithm. *Lecture Notes in Computer Science 4372* (2007), 184–195.

[141] **Putnam, R. T., Sleeman, D., Baxter, J. A., and Kuspa, L. K.** A summary of misconceptions of high school basic programmers. *Journal of Educational Computing Research 2*, 4 (1986), 459–472.

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

[143] **Qian, Y., Hambrusch, S., Yadav, A., Gretter, S., and Li, Y.** Teachers' perceptions of student misconceptions in introductory programming. *Journal of Educational Computing Research 58*, 2 (2020), 364–397.

[144] **Qian, Y., and Lehman, J.** Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education 18*, 1 (Oct. 2017).

[145] **Rader, C., Hakkarinen, D., Moskal, B. M., and Hellman, K.** Exploring the appeal of socially relevant computing: Are students interested in socially relevant problems? In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2011), SIGCSE '11, Association for Computing Machinery, p. 423–428.

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

[147] **Reisner, P.** Human Factors Studies of Database Query Languages: A Survey and Assessment. *ACM Computing Surveys 13*, 1 (jan 1981), 13–31.

[148] **Reisner, P.** Measurement of SQL Problems and Progress. In *Application Development Systems* (1986), Springer-Verlag Tokyo, p. 165–187.

[149] **Rich, K. M., Strickland, C., Andrew Binkowski, T., Moran, C., and Franklin, D.** K–8 learning trajectories derived from research literature: Sequence, repetition, conditionals. *ACM Inroads 9*, 1 (2018), 46–55.

[150] **Rich, L., Perry, H., and Guzdial, M.** A CS1 course designed to address interests of women. In *Proceedings of the 35th SIGCSE technical symposium on Computer Science Education* (2004), ACM New York, NY, USA, pp. 190–194.

[151] **Robins, A., Rountree, J., and Rountree, N.** Learning and teaching programming: A review and discussion. *Computer Science Education 13*, 2 (2003), 137–172.

[152] **Sadiq, S., Orlowska, M., Sadiq, W., and Lin, J.** SQLator - An online SQL learning workbench. In *Proceedings of the 9th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)* (2004), pp. 223–227.

[153] **Scherz, Z., Goldberg, D., and Fund, Z.** Cognitive Implications of Learning Prolog — Mistakes and Misconceptions. *Journal of Educational Computing Research 6*, 1 (1990), 89–110.

[154] **Schnotz, W., and Bannert, M.** Construction and interference in learning from multiple representation. *Learning and Instruction 13*, 2 (2003), 141–156.

[155] **Scholtz, J., and Wiedenbeck, S.** Learning second and subsequent programming languages: A problem of transfer. *International Journal of Human-Computer Interaction 2*, 1 (1990), 51–72.

[156] **Sen, J., Lei, C., Quamar, A., Özcan, F., Efthymiou, V., Dalmia, A., Stager, G., Mittal, A., Saha, D., and Sankaranarayanan, K.** Athena++ natural language querying for complex nested sql queries. *Proceedings of the VLDB Endowment 13*, 12 (2020), 2747–2759.

[157] **Shin, S.** Structured Query Language Learning: Concept Map-Based Instruction Based on Cognitive Load Theory. *IEEE Access 8* (2020), 100095–100110.

[158] **Siau, K. L., Chan, H. C., and Wei, K. K.** 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.

[159] **Simon, Lopez, M., Sutton, K., and Clear, T.** Surely We Must Learn to Read before We Learn to Write! In *Eleventh Australasian Computing Education Conference* (Jan 2009), vol. 95.

[160] **Simon, and Snowdon, S.** Explaining program code: giving students the answer helps - but only just. In *Proceedings of the seventh international workshop on Computing education research* (Providence Rhode Island USA, Aug 2011), ACM, p. 93–100.

[161] **Skinner, E. A., and Belmont, M. J.** Motivation in the classroom: Reciprocal effects of teacher behavior and student engagement across the school year. *Journal of educational psychology 85*, 4 (1993), 571.

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

[163] **Smith-III, J. P., diSessa, A. A., and Roschelle, J.** Misconceptions reconceived: A constructivist analysis of knowledge in transition. *Journal of the Learning Sciences 3*, 2 (1994), 115–163.

[164] **solid IT**. Db-engines ranking. `https://db-engines.com/en/ranking`, 2023. [Online; accessed 24 August 2023].

[165] **Sorva, J.** The Same but Different: Students' Understandings of Primitive and Object Variables. In *Proceedings of the 8th International Conference on Computing Education Research* (New York, NY, USA, 2008), Koli '08, Association for Computing Machinery, p. 5–15.

[166] **Sorva, J.** *Visual Program Simulation in Introductory Programming Education*. PhD thesis, Aalto University, Espoo, Finland, 2012.

[167] **Sowa, J. F.** Conceptual graphs. *Foundations of Artificial Intelligence 3* (2008), 213–237.

[168] **Soylu, A., Giese, M., Jimenez-Ruiz, E., Vega-Gorgojo, G., and Horrocks, I.** Experiencing OptiqueVQS: a multi-paradigm and ontology-based visual query system for end users. *Universal Access in the Information Society 15*, 1 (2016), 129–152.

[169] **Streveler, R. A., Olds, B. M., Miller, R. L., and Nelson, M. A.** Using a Delphi study to identity the most difficult concepts for students to master in thermal and transport science. In *ASEE Annual Conference Proceedings* (2003), pp. 4447–4454.

[170] **Sweller, J.** Cognitive load during problem solving: Effects on learning. *Cognitive Science 12*, 2 (1988), 257–285.

[171] **Swidan, A., Hermans, F., and Smit, M.** Programming misconceptions for school students. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (New York, NY, USA, 2018), ICER '18, Association for Computing Machinery, p. 151–159.

[172] **Taipalus, T.** Explaining Causes behind SQL Query Formulation Errors. *Proceedings - Frontiers in Education Conference, FIE 2020-October* (2020), 1–9.

[173] **Taipalus, T.** The effects of database complexity on SQL query formulation. *Journal of Systems and Software 165* (2020), 110576.

[174] **Taipalus, T., and Grahn, H.** NewSQL Database Management System Compiler Errors: Effectiveness and Usefulness. *International Journal of Human–Computer Interaction* (Aug 2022), 1–12.

[175] **Taipalus, T., and Grahn, H.** Framework for SQL Error Message Design: A Data-Driven Approach. *ACM Transactions on Software Engineering and Methodology* (Jul 2023), 3607180.

[176] **Taipalus, T., Grahn, H., and Ghanbari, H.** Error messages in relational database management systems: A comparison of effectiveness, usefulness, and user confidence. *Journal of Systems and Software 181* (2021), 111034.

[177] **Taipalus, T., Miedema, D., and Aivaloglou, E.** Engaging Databases for Data Systems Education. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland, 2023), pp. 334–340.

[178] **Taipalus, T., and Perälä, P.** 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)* (2019), pp. 198–203.

[179] **Taipalus, T., and Seppänen, V.** SQL Education: A Systematic Mapping Study and Future Research Agenda. *ACM Transactions on Computing Education 20*, 3 (Aug. 2020).

[180] **Taipalus, T., Siponen, M., and Vartiainen, T.** Errors and Complications in SQL Query Formulation. *ACM Transactions on Computing Education 18*, 3 (2018).

[181] **Tamir, P.** Some issues related to the use of justifications to multiple-choice answers. *Journal of Biological Education 23*, 4 (1989), 285–292.

[182] **Tew, A. E., and Guzdial, M.** Developing a validated assessment of fundamental CS1 concepts. In *Proceedings of the 41st ACM technical symposium on Computer science education* (Milwaukee Wisconsin USA, Mar 2010), ACM, p. 97–101.

[183] **Thalheim, B.** Visual SQL - high-quality ER-Based query treatment. *Lecture Notes in Computer Science* (2003).

[184] **Thalheim, B.** Visual sql: towards er-based object-relational database querying. In *International Conference on Conceptual Modeling* (2008), Springer, pp. 520–521.

[185] **Tippett, C. D.** Refutation Text In Science Education: A Review Of Two Decades Of Research. *International Journal of Science and Mathematics Education 8*, 6 (2010), 951–970.

[186] **Tory, M., and Moller, T.** Human factors in visualization research. *IEEE Transactions on Visualization and Computer Graphics 10*, 1 (2004), 72–84.

[187] **Tshukudu, E., and Cutts, Q.** Understanding conceptual transfer for students learning new programming languages. In *Proceedings of the 2020 ACM conference on International Computing Education Research* (2020), pp. 227–237.

[188] **Tshukudu, E., Cutts, Q., Goletti, O., Swidan, A., and Hermans, F.** Teachers' views and experiences on teaching second and subsequent programming languages. In *Proceedings of the 17th ACM Conference on International Computing Education Research* (2021), pp. 294–305.

[189] **Venables, A., Tan, G., and Lister, R.** A closer look at tracing, explaining and code writing skills in the novice programmer. *ICER'09 - Proceedings of the 2009 ACM Workshop on International Computing Education Research* (2009), 117–128.

[190] **Wagner, P. J., Shoop, E., and Carlis, J. V.** Using scientific data to teach a database systems course. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 2003), ACM, pp. 224–228.

[191] **Wahidah, N., and Saptono, S.** The development of three tier multiple choice test to explore junior high school students ' scientific literacy misconceptions. *Journal of Innovative Science Education 8*, 2 (2019), 190–198.

[192] **Ware, C., Purchase, H., Colpoys, L., and McGill, M.** Cognitive measurements of graph aesthetics. *Information Visualization 1*, 2 (2002), 103–110.

[193] **Welty, C.** Correcting user errors in SQL. *Int. J. Man-Machine Studies 22* (1985), 463–477.

[194] **Welty, C., and Stemple, D. W.** Human factors comparison of a procedural and a non-procedural query language. *ACM Transactions on Database Systems 6*, 4 (dec 1981), 626–649.

[195] **Whalley, J., and Kasto, N.** 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.

[196] **Wittie, L., Kurdia, A., Peng, J., Kelly, J., and Huggard, M.** Developing a Concept Inventory for Computer Science 2: What should it focus on and what makes it challenging? In *2020 IEEE Frontiers in Education Conference (FIE)* (2020), pp. 1–5.

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

[198] **Yarosh, S., and Guzdial, M.** Narrating data structures: The role of context in CS2. *Journal on Educational Resources in Computing (JERIC) 7*, 4 (2008), 1–20.

[199] **YeckehZaare, I., Resnick, P., and Ericson, B.** A spaced, interleaved retrieval practice tool that is motivating and effective. In *Proceedings of the 15th ACM Conference on International Computing Education Research* (2019), p. 71–79.

[200] **Yen, C.-Z., Wu, P.-H., and Lin, C.-F.** Analysis of experts' and novices' thinking process in program debugging. In *International Conference on ICT in Teaching and Learning* (2012), Springer, pp. 122–134.

[201] **Yousuf, M. I.** Using experts' opinions through Delphi technique. *Practical Assessment, Research and Evaluation 12*, 4 (2007), 1–7.

[202] **Yue, K.-B.** Using a semi-realistic database to support a database course. *Journal of Information Systems Education 24*, 4 (2013), 327–336.

[203] **Yung, H. I., and Paas, F.** Effects of computer-based visual representation on mathematics learning and cognitive load. *Educational Technology and Society 18*, 4 (2015), 70–77.

[204] **Žanko, Ž., Mladenović, M., and Boljat, I.** Misconceptions about variables at the K-12 level. *Education and Information Technologies 24*, 2 (2019), 1251–1268.

[205] **Zhi, R., Chi, M., Barnes, T., and Price, T. W.** Evaluating the effectiveness of parsons problems for block-based programming. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (2019), pp. 51–59.

[206] **Zloof, M. M.** Query by Example. In *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition* (New York, 1975), IBM T.J. Watson Research Center, pp. 431–438.

Summary

The Structured Query Language (SQL) is ubiquitous, both in industry and education. Therefore, it is important to educate our students in such a way that they can effectively work with SQL. Introductory database courses, which cover query languages such as SQL, are an essential part of most Computer Science bachelor degrees.

While the syntax of SQL is relatively straightforward in comparison to most programming languages, it is not a trivial language to learn. SQL is highly expressive, but as a result, it also has a high complexity. Previous work has already found that there are aspects of SQL that lead to high complexity: strict syntax rules, the cognitive load associated with translating the language, lack of correct and complete error messages, and interference of existing knowledge. All these factors contribute to the difficulty of learning SQL.

Although there has been some research on why SQL is difficult, much work remains to be done to make SQL learning more accessible. In this dissertation, I worked on extending the research on students' SQL struggles. I have studied two directions: understanding student struggles, and ways to support students. My work on understanding student struggles contains four studies:

1. Examining the problem-solving process quantitatively, through analysis of homework submissions. We analysed the complexity of students' solutions through four perspectives: correctness, execution order, edit distance, and query intricacy. We found that students mismanage complexity by writing overly elaborate queries. We call this self-inflicted query complexity.

2. Examining the problem-solving process qualitatively, through a think-aloud study. With the data from this study we were able to identify twelve misconceptions for SQL in four categories: misconceptions based on previous course knowledge, generalisation-based misconceptions, language-based misconceptions, and misconceptions due to an incomplete or incorrect mental model.

3. Examining SQL experts' (educators, researchers and practitioners) perspectives on student errors, through a Delphi-like study. In a two-round process, our experts suggested and voted on potential (additional) underlying causes for student errors. This led to a set of hypotheses per error, instead of the individual misconceptions we had in the previous paper.

4. Exploring the representativeness of the previously identified misconceptions, through an online multiple-choice questionnaire. We found that all misconceptions within the questionnaire are held to some extent. Additionally, from the textual explanations and certainty scores that students provided, we have been able to uncover previously unidentified areas of struggle, allowing us to identify new misconceptions.

Based on these findings, we get a clearer insight into potential areas of struggle for our students. With this information, we can then design instructional approaches that appropriately address these difficulties. During the work on the above-mentioned studies, I gained insight into various ways in which we could support our students. This led to the following two works of support:

1. We have learned that low expressive ease and high cognitive load characterise SQL usage. Research shows that visual representations can assist learners by significantly lowering this burden. We developed a new type of visual representation that displays the query as written in SQL by the user. The aim of the system is to support query formulation, and in turn improve the SQL writing proficiency of its users.
2. We have learned that toy examples are used to introduce key concepts related to database design and to illustrate querying. However, little is known about what makes good example databases. Therefore, we explored students' system design reports to learn how to design engaging databases for education. We identified factors that would make data systems education more engaging: six factors regarding engaging domains, five factors for engaging complexity, and three factors for enough data. The main factor for domain-related engagement was *personal interest*, the main factor for complexity engagement was *matching information requirements*, and the main factor for engaging amounts of data was *realistic data*.

Curriculum Vitæ

Daphne Miedema was born on August 13th 1993 in Woerden, The Netherlands. She completed a Bachelor's degree in Web Science at Eindhoven University of Technology (TU/e) in 2015 and remained there for her double Master of Science degree in both Computer Science and Engineering and Human-Technology Interaction. She completed these degrees in 2019 with a thesis on successful interaction between humans and databases (grades: 9.0 and 8.5). Afterward, she started her PhD program in the Databases Group at TU/e under the supervision of prof. dr. George Fletcher and dr. Efthimia Aivaloglou.

During her PhD, Daphne has been actively involved in both research and teaching activities. She was a lecturer and/or instructor for the courses 2IAB0 Data Analytics for Engineers, 2ID50 Data Modeling and Databases, and 2AMD20 Knowledge Engineering. On the research side, Daphne spearheaded the setup of the DataEd workshop at the SIGMOD conference, a premier conference for Data Systems research. DataEd has seen two highly successful iterations so far. Additionally, she has served as a reviewer for both the SIGCSE conference and the Computer Science Education journal.

Daphne contributed to 16 papers during her PhD, of which she was the first author for nine. She also received two best paper nominations. The first is for her first-author work *Identifying SQL Misconceptions of Novices: Findings from a Think-Aloud Study* at the ICER conference in 2021. The second is on her work as a second author for the paper *Engaging Databases for Data Systems Education* at the ITiCSE conference in 2023.

Acknowledgments

The PhD journey is done, and what a journey it was. Over the past four years I have learnt a lot, about writing, presenting, networking, but also about myself. This experience has helped me grow into the curious, independent researcher that I am today. Returning to the quote that I opened this dissertation with:

> "Man, unlike any other thing organic or inorganic in the universe, grows beyond his work, walks up the stairs of his concepts, and emerges ahead of his accomplishments."

I really believe that this quote reflects the PhD journey, both on a personal, as well as on a conceptual level. The learning process may sometimes feel like being thrown in cold water, but as I recently learned on the Falls of Bruar, cold water may have a lot of wonderful things to offer.

My first and foremost, deepest gratitude goes to my academic parents and promotores prof. dr. George Fletcher and dr. Fenia Aivaloglou. George, thank you for your mentorship and your sincere kindness. Your questions and deep reflections have, and always will, provide valuable food for thought. Fenia, thank you for always being there for me, even on short notice. Thank you for the dinners, the drinks and the calls.

I'd like to thank my committee members for their valuable feedback on this document: prof. dr. Esther Ventura-Medina from Eindhoven University of Technology, prof. dr. ir. Felienne Hermans from the Vrije Universiteit Amsterdam, prof. Andrew Petersen from University of Toronto Mississauga, dr. Sourav Bhowmick from Nanyang Technological University and prof. dr. Scherzinger from University of Passau. A special thanks to Felienne for providing me with career advice.

Massive thanks to my collaborators and co-authors for the wonderful work we were able to do together: Toni Taipalus, Michael Liut, Sourav Bhowmick, Nick Yakovets, Alex Odynchuk and Dmitrii Orlov. Together, we can achieve more.

I also would like to thank all people who have worked with George, Fenia and myself on making DataEd a success. It is wonderful to see the community grow and mature. Thank you to our authors, reviewers, advisory board members, keynote speakers, sponsors, reflection co-authors and of course to all attendees!

At TU/e I also worked with many wonderful colleagues. A big thank you to all secretaries: José, Ine, and Mariela for your support throughout the years. Your genuine interest and motherly concern is heartwarming.

Thank you to all former and current officemates: Yorick Spenrath, Gabriel Cantanhede, Azadeh Mozafari Mehr, Hamid Sharivari Joghan, Akrati Saxena, Fang Lyu, Ziwei Jin, Daan

de Graaf and Larissa Capobianco Shimomura. Thank you for sharing your thoughts, both on work and on everything outside it. Sharing an office is not always ideal, but it definitely brings more fun into the workplace.

Thank you to all other colleagues of the Database group: Bram van de Wall, Daniele Bonetta, Jens d'Hondt, Kaijie Zhu, Nick Yakovets, Odysseas Papapetrou, Robert Brijder, Seperh Sadoughi, Stanley Clark, Thomas Mulder, Wieger Punter, Wilco van Leeuwen and Yuanjin Wu. I really appreciated our Covid-times stand-ups, being together virtually during those lonely days. Thank you for celebrating the good and talking out the bad.

I am also grateful for all the colleagues who were in for hallway talks, lunch, coffee, beer or dinner over the years: Alvaro, Astrid, Bilge, Bram, Danil, David, Dominique, Erik, Ghada, Hannah, Hilde, Israel, Jakub, Jan, Linhao, Loek, Maryam, Mitchel, Mykola, Pieter, Prabhant, Rachid, Rianne, Shiwei, Sibylle, Simon & Simon, Tristan, Wouter, Xin, Yorick, Zeno, and many others. The community that we built as colleagues within the Data Science domain is something I will take with me wherever I may end up next.

Finally, I'd like to thank my family for their love and support. You are always ready to listen, to celebrate, to commiserate. I appreciate it. Oma, opa Ton, you have inspired me to develop my best self. I'll share the love by passing this on to the next generation. Last, and definitely not least, Dennis, my rock, my shelter. It has been difficult, but easy at the same time. Thank you, for everything. Let's go on an adventure.

Daphne Eveline Miedema
Eindhoven, November 2023

SIKS dissertations

2016
01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
04 Laurens Rietveld (VUA), Publishing and Consuming Linked Data
05 Evgeny Sherkhonov (UvA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
06 Michel Wilson (TUD), Robust scheduling in an uncertain environment
07 Jeroen de Man (VUA), Measuring and modeling negative emotions for virtual training
08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
09 Archana Nottamkandath (VUA), Trusting Crowdsourced Information on Cultural Artefacts
10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
11 Anne Schuth (UvA), Search Engines that Learn from Their Users
12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
13 Nana Baah Gyan (VUA), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization
15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
16 Guangliang Li (UvA), Socially Intelligent Autonomous Agents that Learn from Human Reward
17 Berend Weel (VUA), Towards Embodied Evolution of Robot Organisms
18 Albert Meroño Peñuela (VUA), Refining Statistical Data on the Web
19 Julia Efremova (TU/e), Mining Social Structures from Genealogical Data
20 Daan Odijk (UvA), Context & Semantics in News & Web Search
21 Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
22 Grace Lewis (VUA), Software Architecture Strategies for Cyber-Foraging Systems
23 Fei Cai (UvA), Query Auto Completion in Information Retrieval
24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach
25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior

26 Dilhan Thilakarathne (VUA), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
30 Ruud Mattheij (TiU), The Eyes Have It
31 Mohammad Khelghati (UT), Deep web content monitoring
32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
33 Peter Bloem (UvA), Single Sample Statistics, exercises in learning from just one example
34 Dennis Schunselaar (TU/e), Configurable Process Trees: Elicitation, Analysis, and Enactment
35 Zhaochun Ren (UvA), Monitoring Social Media: Summarization, Classification and Recommendation
36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
40 Christian Detweiler (TUD), Accounting for Values in Design
41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
42 Spyros Martzoukos (UvA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
44 Thibault Sellam (UvA), Automatic Assistants for Database Exploration
45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
46 Jorge Gallego Perez (UT), Robots to Make you Happy
47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
48 Tanja Buttler (TUD), Collecting Lessons Learned
49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
50 Yan Wang (TiU), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

2017 01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime

02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
05 Mahdieh Shadi (UvA), Collaboration Behavior
06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search
07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly
08 Rob Konijn (VUA), Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text
10 Robby van Delden (UT), (Steering) Interactive Play Behavior
11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
12 Sander Leemans (TU/e), Robust Process Mining with Guarantees
13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology
14 Shoshannah Tekofsky (TiU), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
15 Peter Berck (RUN), Memory-Based Text Correction
16 Aleksandr Chuklin (UvA), Understanding and Modeling Users of Modern Search Engines
17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
18 Ridho Reinanda (UvA), Entity Associations for Search
19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
22 Sara Magliacane (VUA), Logics for causal inference under uncertainty
23 David Graus (UvA), Entities of Interest — Discovery in Digital Traces
24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning
25 Veruska Zamborlini (VUA), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
28 John Klein (VUA), Architecture Practices for Complex Contexts
29 Adel Alhuraibi (TiU), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"
30 Wilma Latuny (TiU), The Power of Facial Expressions

	31	Ben Ruijl (UL), Advances in computational methods for QFT calculations
	32	Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
	33	Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
	34	Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
	35	Martine de Vos (VUA), Interpreting natural science spreadsheets
	36	Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
	37	Alejandro Montes Garcia (TU/e), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
	38	Alex Kayal (TUD), Normative Social Applications
	39	Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
	40	Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
	41	Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
	42	Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
	43	Maaike de Boer (RUN), Semantic Mapping in Video Retrieval
	44	Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
	45	Bas Testerink (UU), Decentralized Runtime Norm Enforcement
	46	Jan Schneider (OU), Sensor-based Learning Support
	47	Jie Yang (TUD), Crowd Knowledge Creation Acceleration
	48	Angel Suarez (OU), Collaborative inquiry-based learning
2018	01	Han van der Aa (VUA), Comparing and Aligning Process Representations
	02	Felix Mannhardt (TU/e), Multi-perspective Process Mining
	03	Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
	04	Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
	05	Hugo Huurdeman (UvA), Supporting the Complex Dynamics of the Information Seeking Process
	06	Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
	07	Jieting Luo (UU), A formal account of opportunism in multi-agent systems
	08	Rick Smetsers (RUN), Advances in Model Learning for Software Systems
	09	Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
	10	Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology
	11	Mahdi Sargolzaei (UvA), Enabling Framework for Service-oriented Collaborative Networks

12 Xixi Lu (TU/e), Using behavioral context in process mining
13 Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
14 Bart Joosten (TiU), Detecting Social Signals with Spatiotemporal Gabor Filters
15 Naser Davarzani (UM), Biomarker discovery in heart failure
16 Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
17 Jianpeng Zhang (TU/e), On Graph Sample Clustering
18 Henriette Nakad (UL), De Notaris en Private Rechtspraak
19 Minh Duc Pham (VUA), Emergent relational schemas for RDF
20 Manxia Liu (RUN), Time and Bayesian Networks
21 Aad Slootmaker (OU), EMERGO: a generic platform for authoring and playing scenario-based serious games
22 Eric Fernandes de Mello Araújo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis
24 Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots
25 Riste Gligorov (VUA), Serious Games in Audio-Visual Collections
26 Roelof Anne Jelle de Vries (UT),Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology
27 Maikel Leemans (TU/e), Hierarchical Process Mining for Scalable Software Analysis
28 Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel
29 Yu Gu (TiU), Emotion Recognition from Mandarin Speech
30 Wouter Beek (VUA), The "K" in "semantic web" stands for "knowledge": scaling semantics to the web

---

2019 01 Rob van Eijk (UL),Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
02 Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty
03 Eduardo Gonzalez Lopez de Murillas (TU/e), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
04 Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
05 Sebastiaan van Zelst (TU/e), Process Mining with Streaming Data
06 Chris Dijkshoorn (VUA), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
07 Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
08 Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes
09 Fahimeh Alizadeh Moghaddam (UvA), Self-adaptation for energy efficiency in software systems
10 Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction

11 Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
12 Jacqueline Heinerman (VUA), Better Together
13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
16 Guangming Li (TU/e), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
17 Ali Hurriyetoglu (RUN),Extracting actionable information from microtexts
18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
21 Cong Liu (TU/e), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
22 Martin van den Berg (VUA),Improving IT Decisions with Enterprise Architecture
23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
24 Anca Dumitrache (VUA), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
25 Emiel van Miltenburg (VUA), Pragmatic factors in (automatic) image description
26 Prince Singh (UT), An Integration Platform for Synchromodal Transport
27 Alessandra Antonaci (OU), The Gamification Design Process applied to (Massive) Open Online Courses
28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
29 Daniel Formolo (VUA), Using virtual agents for simulation and training of social skills in safety-critical circumstances
30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
31 Milan Jelisavcic (VUA), Alive and Kicking: Baby Steps in Robotics
32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games
33 Anil Yaman (TU/e), Evolution of Biologically Inspired Learning in Artificial Neural Networks
34 Negar Ahmadi (TU/e), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
35 Lisa Facey-Shaw (OU), Gamification with digital badges in learning programming
36 Kevin Ackermans (OU), Designing Video-Enhanced Rubrics to Master Complex Skills
37 Jian Fang (TUD), Database Acceleration on FPGAs
38 Akos Kadar (OU), Learning visually grounded and multilingual representations

2020
01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour
02 Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models
03 Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding
04 Maarten van Gompel (RUN), Context as Linguistic Bridges
05 Yulong Pei (TU/e), On local and global structure mining
06 Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support
07 Wim van der Vegt (OU), Towards a software architecture for reusable game components
08 Ali Mirsoleimani (UL),Structured Parallel Programming for Monte Carlo Tree Search
09 Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
10 Alifah Syamsiyah (TU/e), In-database Preprocessing for Process Mining
11 Sepideh Mesbah (TUD), Semantic-Enhanced Training Data Augmentation-Methods for Long-Tail Entity Recognition Models
12 Ward van Breda (VUA), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
13 Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
14 Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases
15 Konstantinos Georgiadis (OU), Smart CAT: Machine Learning for Configurable Assessments in Serious Games
16 Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling
17 Daniele Di Mitri (OU), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences
18 Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems
19 Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems
20 Albert Hankel (VUA), Embedding Green ICT Maturity in Organisations
21 Karine da Silva Miras de Araujo (VUA), Where is the robot?: Life as it could be
22 Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar
23 Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging
24 Lenin da Nóbrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots
25 Xin Du (TU/e), The Uncertainty in Exceptional Model Mining
26 Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer opTimization
27 Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context

	28	Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
	29	Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
	30	Bob Zadok Blok (UL), Creatief, Creatiever, Creatiefst
	31	Gongjin Lan (VUA), Learning better – From Baby to Better
	32	Jason Rhuggenaath (TU/e), Revenue management in online markets: pricing and online advertising
	33	Rick Gilsing (TU/e), Supporting service-dominant business model evaluation in the context of business model innovation
	34	Anna Bon (UM), Intervention or Collaboration? Redesigning Information and Communication Technologies for Development
	35	Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software Production
2021	01	Francisco Xavier Dos Santos Fonseca (TUD),Location-based Games for Social Interaction in Public Space
	02	Rijk Mercuur (TUD), Simulating Human Routines: Integrating Social Practice Theory in Agent-Based Models
	03	Seyyed Hadi Hashemi (UvA), Modeling Users Interacting with Smart Devices
	04	Ioana Jivet (OU), The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning
	05	Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous Systems
	06	Daniel Davison (UT), "Hey robot, what do you think?" How children learn with a social robot
	07	Armel Lefebvre (UU), Research data management for open science
	08	Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Programming on Computational Thinking
	09	Cristina Zaga (UT), The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Children's Collaboration Through Play
	10	Quinten Meertens (UvA), Misclassification Bias in Statistical Learning
	11	Anne van Rossum (UL), Nonparametric Bayesian Methods in Robotic Vision
	12	Lei Pi (UL), External Knowledge Absorption in Chinese SMEs
	13	Bob R. Schadenberg (UT), Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning
	14	Negin Samaeemofrad (UL), Business Incubators: The Impact of Their Support
	15	Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
	16	Esam A. H. Ghaleb (UM), Bimodal emotion recognition from audio-visual cues
	17	Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks
	18	Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks
	19	Roberto Verdecchia (VUA), Architectural Technical Debt: Identification and Management
	20	Masoud Mansoury (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems

21 Pedro Thiago Timbó Holanda (CWI), Progressive Indexes
22 Sihang Qiu (TUD), Conversational Crowdsourcing
23 Hugo Manuel Proença (UL), Robust rules for prediction and description
24 Kaijie Zhu (TU/e), On Efficient Temporal Subgraph Query Processing
25 Eoin Martino Grua (VUA), The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications
26 Benno Kruit (CWI/VUA), Reading the Grid: Extending Knowledge Bases from Human-readable Tables
27 Jelte van Waterschoot (UT), Personalized and Personal Conversations: Designing Agents Who Want to Connect With You
28 Christoph Selig (UL), Understanding the Heterogeneity of Corporate Entrepreneurship Programs

---

2022 01 Judith van Stegeren (UT), Flavor text generation for role-playing video games
02 Paulo da Costa (TU/e), Data-driven Prognostics and Logistics Optimisation: A Deep Learning Journey
03 Ali el Hassouni (VUA), A Model A Day Keeps The Doctor Away: Reinforcement Learning For Personalized Healthcare
04 Ünal Aksu (UU), A Cross-Organizational Process Mining Framework
05 Shiwei Liu (TU/e), Sparse Neural Network Training with In-Time Over-Parameterization
06 Reza Refaei Afshar (TU/e), Machine Learning for Ad Publishers in Real Time Bidding
07 Sambit Praharaj (OU), Measuring the Unmeasurable? Towards Automatic Co-located Collaboration Analytics
08 Maikel L. van Eck (TU/e), Process Mining for Smart Product Design
09 Oana Andreea Inel (VUA), Understanding Events: A Diversity-driven Human-Machine Approach
10 Felipe Moraes Gomes (TUD), Examining the Effectiveness of Collaborative Search Engines
11 Mirjam de Haas (UT), Staying engaged in child-robot interaction, a quantitative approach to studying preschoolers' engagement with robots and tasks during second-language tutoring
12 Guanyi Chen (UU), Computational Generation of Chinese Noun Phrases
13 Xander Wilcke (VUA), Machine Learning on Multimodal Knowledge Graphs: Opportunities, Challenges, and Methods for Learning on Real-World Heterogeneous and Spatially-Oriented Knowledge
14 Michiel Overeem (UU), Evolution of Low-Code Platforms
15 Jelmer Jan Koorn (UU), Work in Process: Unearthing Meaning using Process Mining
16 Pieter Gijsbers (TU/e), Systems for AutoML Research
17 Laura van der Lubbe (VUA), Empowering vulnerable people with serious games and gamification
18 Paris Mavromoustakos Blom (TiU), Player Affect Modelling and Video Game Personalisation
19 Bilge Yigit Ozkan (UU), Cybersecurity Maturity Assessment and Standardisation

20 Fakhra Jabeen (VUA), Dark Side of the Digital Media - Computational Analysis of Negative Human Behaviors on Social Media
21 Seethu Mariyam Christopher (UM), Intelligent Toys for Physical and Cognitive Assessments
22 Alexandra Sierra Rativa (TiU), Virtual Character Design and its potential to foster Empathy, Immersion, and Collaboration Skills in Video Games and Virtual Reality Simulations
23 Ilir Kola (TUD), Enabling Social Situation Awareness in Support Agents
24 Samaneh Heidari (UU), Agents with Social Norms and Values - A framework for agent based social simulations with social norms and personal values
25 Anna L.D. Latour (UL), Optimal decision-making under constraints and uncertainty
26 Anne Dirkson (UL), Knowledge Discovery from Patient Forums: Gaining novel medical insights from patient experiences
27 Christos Athanasiadis (UM), Emotion-aware cross-modal domain adaptation in video sequences
28 Onuralp Ulusoy (UU), Privacy in Collaborative Systems
29 Jan Kolkmeier (UT), From Head Transform to Mind Transplant: Social Interactions in Mixed Reality
30 Dean De Leo (CWI), Analysis of Dynamic Graphs on Sparse Arrays
31 Konstantinos Traganos (TU/e), Tackling Complexity in Smart Manufacturing with Advanced Manufacturing Process Management
32 Cezara Pastrav (UU), Social simulation for socio-ecological systems
33 Brinn Hekkelman (CWI/TUD), Fair Mechanisms for Smart Grid Congestion Management
34 Nimat Ullah (VUA), Mind Your Behaviour: Computational Modelling of Emotion & Desire Regulation for Behaviour Change
35 Mike E.U. Ligthart (VUA), Shaping the Child-Robot Relationship: Interaction Design Patterns for a Sustainable Interaction

---

2023 01 Bojan Simoski (VUA), Untangling the Puzzle of Digital Health Interventions
02 Mariana Rachel Dias da Silva (TiU), Grounded or in flight? What our bodies can tell us about the whereabouts of our thoughts
03 Shabnam Najafian (TUD), User Modeling for Privacy-preserving Explanations in Group Recommendations
04 Gineke Wiggers (UL), The Relevance of Impact: bibliometric-enhanced legal information retrieval
05 Anton Bouter (CWI), Optimal Mixing Evolutionary Algorithms for Large-Scale Real-Valued Optimization, Including Real-World Medical Applications
06 António Pereira Barata (UL), Reliable and Fair Machine Learning for Risk Assessment
07 Tianjin Huang (TU/e), The Roles of Adversarial Examples on Trustworthiness of Deep Learning
08 Lu Yin (TU/e), Knowledge Elicitation using Psychometric Learning
09 Xu Wang (VUA), Scientific Dataset Recommendation with Semantic Techniques

10 Dennis J.N.J. Soemers (UM), Learning State-Action Features for General Game Playing
11 Fawad Taj (VUA), Towards Motivating Machines: Computational Modeling of the Mechanism of Actions for Effective Digital Health Behavior Change Applications
12 Tessel Bogaard (VUA), Using Metadata to Understand Search Behavior in Digital Libraries
13 Injy Sarhan (UU), Open Information Extraction for Knowledge Representation
14 Selma Čaušević (TUD), Energy resilience through self-organization
15 Alvaro Henrique Chaim Correia (TU/e), Insights on Learning Tractable Probabilistic Graphical Models
16 Peter Blomsma (TiU), Building Embodied Conversational Agents: Observations on human nonverbal behaviour as a resource for the development of artificial characters
17 Meike Nauta (UT), Explainable AI and Interpretable Computer Vision – From Oversight to Insight
18 Gustavo Penha (TUD), Designing and Diagnosing Models for Conversational Search and Recommendation
19 George Aalbers (TiU), Digital Traces of the Mind: Using Smartphones to Capture Signals of Well-Being in Individuals
20 Arkadiy Dushatskiy (TUD), Expensive Optimization with Model-Based Evolutionary Algorithms applied to Medical Image Segmentation using Deep Learning
21 Gerrit Jan de Bruin (UL), Network Analysis Methods for Smart Inspection in the Transport Domain
22 Alireza Shojaifar (UU), Volitional Cybersecurity
23 Theo Theunissen (UU), Documentation in Continuous Software Development
24 Agathe Balayn (TUD), Practices Towards Hazardous Failure Diagnosis in Machine Learning
25 Jurian Baas (UU), Entity Resolution on Historical Knowledge Graphs
26 Loek Tonnaer (TU/e), Linearly Symmetry-Based Disentangled Representations and their Out-of-Distribution Behaviour
27 Ghada Sokar (TU/e), Learning Continually Under Changing Data Distributions
28 Floris den Hengst (VUA), Learning to Behave: Reinforcement Learning in Human Contexts
29 Tim Draws (TUD), Understanding Viewpoint Biases in Web Search Results

2024 01 **Daphne Miedema** (TU/e), On Learning SQL: Disentangling concepts in data systems education