SQL Puzzles: Evaluating Micro Parsons Problems With Different Types of Feedback as Practice for Novice Programmers

Zihan Wu
ziwu@umich.edu
University of Michigan
Ann Arbor, Michigan, United States

Barbara J. Ericson
barbarer@umich.edu
University of Michigan
Ann Arbor, Michigan, United States

ABSTRACT

This paper investigates using micro Parsons problems as a novel practice approach for learning Structured Query Language (SQL). In micro Parsons problems learners arrange predefined code fragments to form a SQL statement instead of typing the code. SQL is a standard language for working with relational databases. Targeting beginner-level SQL statements, we evaluated the efficacy of micro Parsons problems with block-based feedback and execution-based feedback compared to traditional text-entry problems. To delve into learners' experiences and preferences for the three problem types, we conducted a within-subjects think-aloud study with 12 participants. We found that learners reported very different preferences. Factors they considered included perceived learning, task authenticity, and prior knowledge. Next, we conducted two between-subjects classroom studies to evaluate the effectiveness of micro Parsons problems with different feedback types versus text-entry problems for SQL practice. We found that learners who practiced by solving Parsons problems with block-based feedback had a significantly higher learning gain than those who practiced with traditional text-entry problems.

CCS CONCEPTS

• **Human-centered computing** → *Interactive systems and tools*; • **Applied computing** → **Interactive learning environments**; • **Social and professional topics** → **Computing education**.

KEYWORDS

programming puzzle, empirical study, SQL education, learning

ACM Reference Format:
Zihan Wu and Barbara J. Ericson. 2024. SQL Puzzles: Evaluating Micro Parsons Problems With Different Types of Feedback as Practice for Novice Programmers. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24), May 11–16, 2024, Honolulu, HI, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3613904.3641910

CHI '24, May 11–16, 2024, Honolulu, HI, USA

ACM ISBN 979-8-4007-0330-0/24/05
https://doi.org/10.1145/3613904.3641910

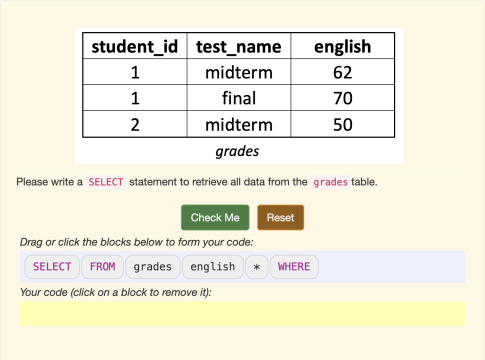
1 INTRODUCTION

Managing databases with Structured Query Language (SQL) is an important skill for students in the field of computer science, software engineering, and information systems [33]. SQL is a domain-specific language used in relational database management systems (RDBMS). The functionality of SQL includes data type definition, data query, data manipulation, and data access control [6]. Because of its importance, SQL is explicitly recommended in many higher education curricula guidelines along with databases. However, teaching SQL is difficult as it requires both subject knowledge and pedagogical skills. A systematic literature review by Taipalus and Seppanen [33] on 89 research papers related to teaching SQL summarized 66 teaching approaches for instructors.

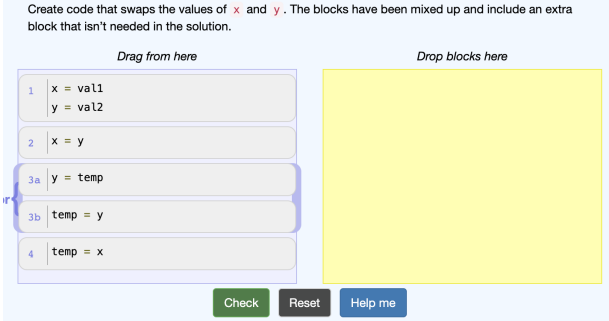
Many interactive environments are available nowadays for learners to practice SQL. However, they mostly involve typing SQL statements [1, 5, 28]. This is a type of whole task, which while authentic, can overload memory and impede learning [32]. Prior work has also designed block-based programming languages for SQL [16, 26, 30], which provide visual interfaces and enable drag-and-drop interactions for beginners, but still contain block banks for all available keywords and code structures for every question. Searching through all available blocks to find the appropriate one can still overwhelm the learners. Completion problems that require learners to complete partial solutions can prevent cognitive overload [32, 34].

In introductory programming education, Parsons problems are a type of completion problem that provide mixed-up solution code in blocks for learners to reconstruct the correct answer by rearranging the order of the blocks [25]. Designed to maximize engagement, model good code, and provide immediate feedback [25], Parsons problems have been found to improve problem-solving efficiency [10] and promote programming pattern acquisition [35], while maintaining equivalent learning gains compared with writing code from scratch [10, 35]. In traditional Parsons problems, each block contains a single or multiple lines of code. A recent study introduced micro Parsons problems, which implemented the idea of Parsons problems at a smaller granularity [40]. A micro Parsons problem provides blocks of code line fragments, and asks learners to reconstruct a single line of code with the blocks. In the context of teaching regex, Wu et al. [40] found that micro Parsons problems encouraged more learners to complete optional practice problems in a MOOC while producing an equivalent learning gain as solving traditional text entry problems. Fig. 1 contrasts a micro Parsons problem and a traditional Parsons problem.

Despite the benefits of Parsons problems, no prior research has explored the use of Parsons problems or micro Parsons problems for practicing SQL. As writing SQL statements requires learners to



(a) Micro Parsons problem with block-based feedback ("Check me" button) for SQL.



(b) Traditional Parsons problem for swapping variable values in Python.

Figure 1: A micro Parsons problem and a traditional Parsons problem. The micro Parsons problems ask learners to rearrange code blocks to form one statement, while the traditional Parsons problems ask learners to rearrange blocks with one or more lines to form a complete program.

assemble table and column names with reserved keywords correctly, micro Parsons problems can potentially be used for practicing SQL. Thus, in this study, we explore the use of micro Parsons problems in SQL in an introductory undergraduate programming course, and explore its effect as practice problems compared with the authentic text-entry task. We investigate two types of feedback for micro Parsons problems: block-based feedback (highlighting incorrect blocks) and execution-based feedback (showing error messages or execution results). With a think-aloud lab study and a classroom study, we answer the following research questions:

- RQ1 - Among micro Parsons problems with block-based feedback, micro Parsons problems with execution-based feedback, and common practice (text-entry problems), what type of practice questions do students prefer, and what are their criteria for preference?
- RQ2 - What are learners' perceived advantages and disadvantages of micro Parsons problems compared with text-entry problems as practice for writing SQL statements?
- RQ3 - What are learners' perceived advantages and disadvantages of block-based feedback and execution-based feedback in micro Parsons problems?
- RQ4 - How do micro Parsons problems affect learners' short-term learning gain and pattern acquisition compared with text-entry problems as practice for writing SQL statements?

By adopting a new way to practice creating SQL statements, studying learners' responses, and evaluating their performance in an authentic classroom setting, our paper contributes the following:

- Adopting micro Parsons problems to create a new type of engaging SQL practice puzzles for novices;
- Understanding how learners interact with micro Parsons problems in the context of SQL, and what affects their learning experience and preferences;
- Collecting learners' perspectives on the input methods of the SQL micro Parsons puzzles as well as different types of feedback, and how they affect learning;
- Providing empirical evidence of the effectiveness of this new type of SQL practice problems in a real classroom setting.

In section two, we review the prior research on learning SQL and existing tools, various types of Parsons problems, and related learning theories. Section three demonstrates the tool interface, as well as the two different types of feedback. Section four details the within-subject think-aloud study for qualitative insights, while section five presents the between-subject classroom studies for quantitative results. Section six discusses our findings with respect to our research questions, and connects the use of micro Parsons problems with prior work on SQL novices' misconceptions. Finally, section seven discusses the limitations and future work, and section eight concludes the paper.

2 RELATED WORK

In this section, we review prior problem types and tools for learners to practice SQL, prior research on Parsons problems, as well as the learning theories that the design of micro Parsons problems draws on.

2.1 Teaching SQL

Structured Query Language (SQL) is used to manage relational databases, and is included in many computer science and software engineering curricula in higher education [3, 24]. SQL skills are crucial for storing and retrieving data. However, prior research has discovered that teaching SQL requires both subject knowledge and pedagogical skills [33].

A systematic literature review on SQL education showed that the most popular topics were student errors and exercise databases [33], instead of creating new tools. Many tools developed for SQL education focused on visualization [12, 20] or automatically grading student answers [17].

There is limited prior work on tools or systems that provide SQL practice or alternative methods for creating SQL statements. SQLRepair by Presler-Marshall et al. automatically fixes learners' SELECT

statements [27]. Brusilovsky et al. designed a more comprehensive SQL adaptive learning system that provides content adaptivity, which navigates learners to different SQL concepts and problem sets based on learners' student models [5]. Similar to most systems, both of these works ask learners to write code from scratch, and do not provide alternative practice methods.

Some systems provide alternative ways for learners to formulate SQL statements. Aisha [2] designed a web-based tool for administering tests in SQL, which provides the complete list of keywords as well as the attributes in the given database, and asks learners to input their answers using a point-and-click method.

Prior work has also developed block-based programming languages for SQL that enable a drag-and-drop input modality, such as SQLsnap [30] (a plugin for Snap! [4]), BlocklySQL [26], and SQheLper [16]. These block-based programming languages for SQL provide a relatively large number of different blocks, such that they can support any SQL statement. For example, SQheLper contains 23 different blocks, and SQLsnap has 30 different blocks [1]. These block-based languages and the work by Aisha [2] offer alternative ways to input SQL statements, and can potentially reduce trivial errors such as misspelling keywords. However, the large number of blocks can increase the cognitive load for novices.

We have not seen any prior work for teaching SQL that uses Parsons problems as a type of SQL practice problem.

2.2 Parsons Problems

While traditional practice problems require learners to write code from scratch based on a problem description, Parsons problems offer an alternative way to provide students with hands-on practice. Parsons problems are a type of programming puzzle that provides mixed-up blocks of solution code along with the problem description, and ask learners to rearrange the code blocks into the correct order [25]. To highlight common mistakes and syntax errors, as well as modeling good practice in writing code, Parsons problems often include distractors, i.e. extra code blocks that are not part of the answer [25].

Parsons problems are often confused with block-based languages [9] because of the similarity in their input modality — drag-and-drop blocks to form code. Block-based programming differs from Parsons problems as they are typically open-ended, and provide a broader range of options to choose from; Parsons problems, however, usually contain a problem statement that limits and defines the question, and have very limited sets of code fragments [9]. The difference in design reflects their different goals. Many block-based programming languages, such as Scratch [22] and Snap! [4], have a focus on creativity [23], and thus provide a wide variety of blocks and slots such that they can support users' creativity needs. Parsons problems, on the other hand, focus on scaffolding problem-solving for given programming problems, and reducing the problem space by limiting the number of blocks.

Researchers have investigated the effect of Parsons problems in different contexts, including block-based programming languages [41] and text-based programming languages [10], in mobile apps and interactive e-books [10], as well as in classrooms [35] and in MOOCs [40]. Ericson et al. [10] found that Parsons problems were a more efficient type of practice problem compared with write code problems, but were just as effective in terms of learning gain. Weinman et al. investigated one variation of Parsons problems that asks learners to fill in some blanks of the Parsons blocks and discovered that it was helpful for programming pattern acquisition [35].

Researchers have explored two types of feedback in Parsons problems. Block-based feedback (or line-based feedback) highlights the code blocks that are incorrect or in the wrong order in learners' solutions. Execution-based feedback runs learners' code and returns the execution results, such as syntax errors from the compiler or interpreter, or the output when the code is executable. A recent literature review on Parsons problems pointed out that there is limited research that studies the feedback types [9]. The only work we were able to locate is from Helminen et al. [14], who conducted a classroom study to compare the effectiveness of the two types of feedback, but did not find differences in terms of learning. However, in the student survey, they found that some students think the execution-based feedback is difficult to understand and did not help them correct their answers. Even in the design space of block-based programming languages, we have not seen prior work comparing different types of feedback for learners. In our work, we aim to generate more insights into learners' perceptions of different feedback types for Parsons problems, which could be potentially helpful for a broader audience that studies block-based programming languages.

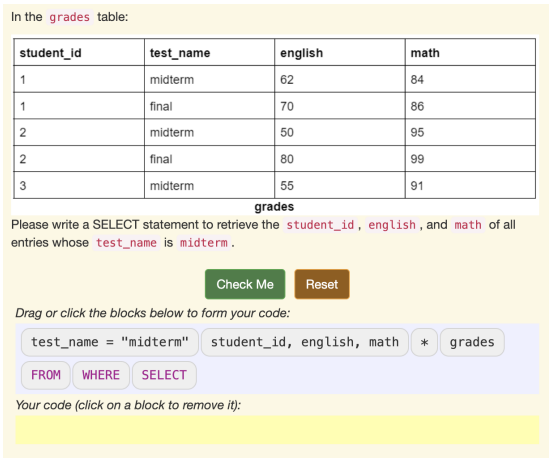
In most variants of Parsons problems, each code block contains one or more lines of code. This limits the ability of Parsons problems to provide practice for learners at a smaller level of granularity. Wu et al. [40] proposed "micro Parsons problems" which ask learners to assemble code fragments in a single line. They implemented micro Parsons problems with execution-based feedback, and evaluated them for learning regex within a MOOC. They discovered that micro Parsons problems significantly reduced the dropout rate from optional practice in a MOOC compared with text-entry problems, and learners in the micro Parsons group performed better on test problems that evaluated their ability to recall the meaning of special regex symbols. However, the empirical evidence of using micro Parsons problems in practice is still limited, and we did not find any prior work that evaluated micro Parsons problems with other programming languages such as SQL, or tested it in classrooms.

2.3 Worked Example Effect and Completion Problem Effect

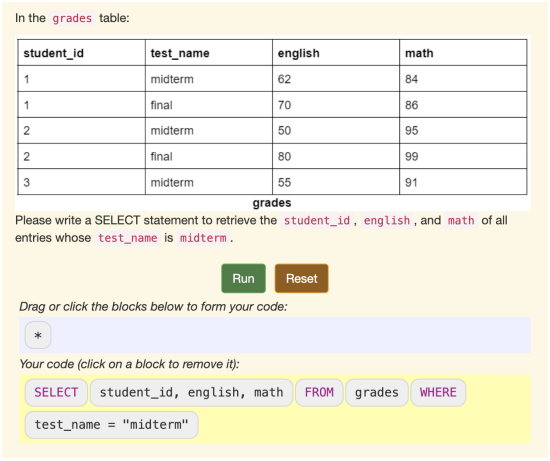
Cognitive load theory (CLT) explains how learning can be negatively affected when learners are provided with information that requires too many cognitive resources to process [8]. CLT identified that the limited capacity of working memory can become a bottleneck for learning if too much information must be processed at once. When instructional materials or activities take up too much working memory, the process of constructing schema will be negatively affected. Thus, one design goal of instructional activities is to avoid cognitive overload for learners.

Both the worked example effect and the completion problem effect originated from CLT. The worked example effect is one of

[1]The number of blocks in SQLsnap was retrieved in December 2023. BlocklySQL did not provide the number of blocks.



(a) Micro Parsons problem with block-based feedback ("Check me" button).



(b) Same micro Parsons problem with execution-based feedback ("Run" button) and with answer filled in.

Figure 2: Micro Parsons SQL problem deployed on Runestone. Before the user requests feedback, the only difference between block-based feedback and execution-based feedback is the text on the left button. Each problem has a problem description, and a micro Parsons input area. The correct answer for this question is demonstrated in (b). Block "*" is a distractor that is not part of the answer.

the most widely studied effects that draw on cognitive load theory [31]. It suggests that asking learners to study worked examples places the focus on understanding problem states and solutions steps while reducing the need for complicated means-ends search for the correct solution.

However, plain worked examples can be less efficient when learners do not carefully study them. Even when they are interleaved with the authentic task, not all learners choose to fully process the worked example prior to attempting the problem on their own. When learners choose to refer to the worked examples during their problem-solving attempts, both the example and the problem they work on are processed in learners' working memory [32], and can potentially result in cognitive overload.

To address this issue, van Merriënboer [34] suggested "completion problems" as an alternative. Compared to the authentic task, *completion problems* provide a partial solution, but still requires learners to complete the solution based on given clues. In the context of programming education, providing learners with a problem description, a code solution to the problem with several blank lines, and asking learners to fill in the blanks, is an intuitive example of completion problems. Learners received a partial solution, but still need to put in cognitive effort to complete it. Parsons problems are also a type of completion problem, as they provide a correct solution in mixed-up order, and require learners to comprehend the code blocks and reconstruct the correct solution.

In theory, completion problems typically decrease extraneous cognitive load by reducing the size of the problem space [32, 34], while forcing learners to interact with the partial solution. Thus, particularly for less experienced learners, Parsons problems can be used to avoid cognitive overload.

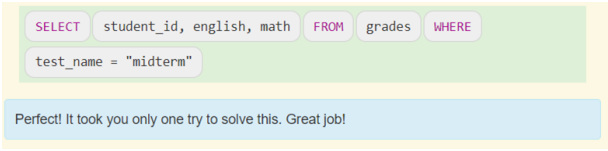
3 SQL PUZZLES: MICRO PARSONS PROBLEMS WITH SQL

We implemented both *block-based* grading (highlighting incorrect code blocks) and *execution-based* grading (executing the code rearranged by students and showing error message/execution result) for SQL in Runestone Academy [21], an online interactive e-book platform.

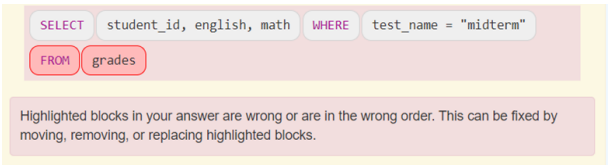
Fig. 2 demonstrates a micro Parsons SQL problem with block-based feedback and a micro Parsons SQL problem with execution-based feedback. A micro Parsons problem contains a problem description, a source area, and a solution area. While traditional Parsons problems usually contain blocks with one or more statements, the answer to introductory-level SQL practices is often limited to one statement. Thus, we adopted micro Parsons problems which focus on the practice of a single statement. Learners can drag or click the blocks in the source area to move them to the solution area. Different from the explanation of special symbols in regex in [40], the blocks in micro Parsons SQL problems do not contain explanations, and are not reusable, as SQL does not usually directly repeat code pieces multiple times as regex (e.g. "\w+" is used repeatedly in "(\w + \.) + (\w+)" for matching a simple URL).

While prior work on traditional Parsons problems has explored block-based feedback and execution-based feedback [14], as of the time of our work, the only existing work on micro Parsons problems only used execution-based feedback in regex [40]. To collect insights on both block-based feedback and execution-based feedback in micro Parsons problems, we implemented micro Parsons problems that support either type of feedback.

When students finish assembling the code blocks, they can click the "Check Me" button (block-based feedback) or "Run" (execution-based feedback) button to request feedback. Then, an area with feedback (see fig. 3 and fig. 4) will appear.



(a) Block-based feedback when the answer is correct.



(b) Block-based feedback when the answer is incorrect.

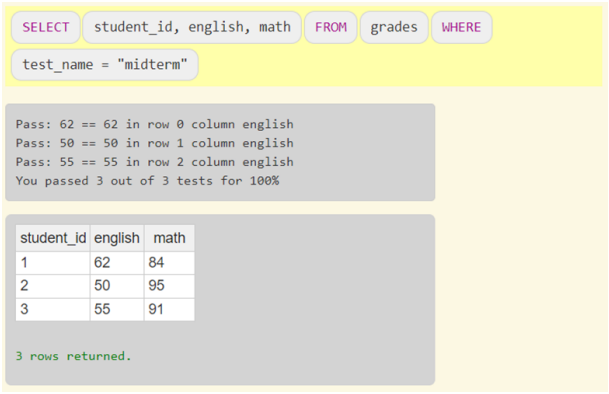
Figure 3: Block-based feedback for the micro Parsons problem in fig. 2(a) when the answer is correct (a) and incorrect (b).

Fig. 3 demonstrates the block-based feedback when the answer is correct (a) and incorrect (b). There are two common algorithms used to provide block-based feedback: longest common subsequence (LCS) and first-incorrect. The LCS method [10] calculates the longest common subsequence between the student's answer and the correct answer, and highlights all blocks in the student's answer that are not in the LCS, whether they are misplaced or are not part of the correct solution. The first-incorrect [29] algorithm highlights the first incorrectly placed block in the learner's solution. In this work, we followed Runestone's existing feedback method for traditional Parsons problems, and implemented the block-based feedback based on LCS. Blocks that are not part of the LCS are highlighted in red.

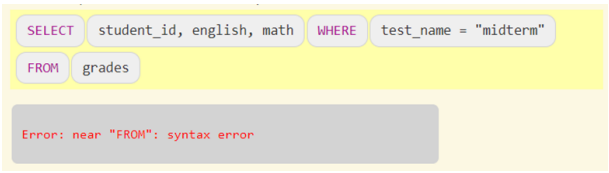
Fig. 4 demonstrates execution-based feedback when learners' answer is correct (a) and incorrect (b). The execution-based feedback for micro Parsons SQL problems consists of unit test results on the top as well as execution results on the bottom. It was implemented to be consistent with the feedback provided for text-entry SQL problems in Runestone. With the support of SQL-js, a JavaScript library that is SQLite compiled to WebAssembly, the system is able to execute students' code in the browser. When the code can be successfully executed, the system displays the data retrieved by the statement (a); When a syntax error is detected in the code or an error occurs while executing the statement, the system will show the error message from SQLite (b). If the tested statement is not SELECT and thus does not return any data (e.g. UPDATE), the instructor can also set the system to visualize the full table to display any change.

4 WITHIN-SUBJECT THINK-ALOUD STUDY

To understand how learners perceive micro Parsons problems with block-based feedback, micro Parsons problems with execution-based feedback, and traditional text-entry problems, we conducted a think-aloud study with a counterbalanced design with 12 participants.



(a) Execution-based feedback when the code is executed without error.



(b) Execution-based feedback when SQLite returns error messages.

Figure 4: Execution-based feedback for the micro Parsons problem in fig. 2 (b) when code can be executed successfully (a) and when there was a syntax error (b).

4.1 Methods

With IRB permission, we sent out a recruitment message as an announcement in the learning management system of an undergraduate data-oriented programming course, which includes basic SQL concepts. The course briefly covers basic SELECT, UPDATE, and JOIN in one week. Both students who had taken the course in the previous semester (learned the SQL concepts at least two months before the recruitment, and were unlikely to be taking advanced SQL courses) and students who were currently taking the course (had not yet started on SQL) received the message through email. We specifically asked for participants who understand the structure of relational databases (tables) and basic SQL statements (SELECT). Twelve participants signed up for the study within the recruitment time range and gave consent.

The think-aloud study was conducted remotely through web conferencing software. Prior to the task, learners completed a pre-task survey, which included basic demographic information. The average age of the participants was 20.75. Seven participants identified as female, four identified as male, and one identified as non-binary. To understand how well the participants of the think-aloud study represent introductory CS learners in general, we also included a 5-point Likert scale six-question self-efficacy survey for CS, which was first developed by Wiebe et al. [38] and revised to a shorter version by Wiggins et al. [39].

To make sure all participants were novices in SQL, we included a self-evaluation survey for participants to self-report their proficiency in SELECT, UPDATE, and JOIN, three key concepts included

in the think-aloud study. No participant reported that they were confident in using UPDATE and JOIN in complex programs, and only one participant reported that they were confident in using SELECT in complex programs.

After the pre-task survey, learners completed a short tutorial on the tool that contained three types of problems. For each problem type, we provided a 30-second video demonstrating how to interact with the tool, construct an answer, and interpret the feedback. We also provided a sample question for learners to gain familiarity with the problem type.

Then, learners were asked to solve 6 SQL problems. We created three versions for each problem: micro Parsons problems with block-based feedback (PB), micro Parsons problems with execution-based feedback (PE), and traditional text-entry problems (TE). For each problem, all three versions had identical problem descriptions. The PB and PE versions had the same set of randomized blocks, and the PE and TE had the same test cases for execution-based feedback.

Table 1: Counterbalanced design of think-aloud study

Group	Practice Problem Type
A	TE - PB - PE - TE - PB - PE
B	TE - PE - PB - TE - PE - PB
C	PB - TE - PE - PB - TE - PE
D	PB - PE - TE - PB - PE - TE
E	PE - TE - PB - PE - TE - PB
F	PE - PB - TE - PE - PB - TE

To reduce the ordering effect, we randomly assigned 12 participants into six groups (see table 1). Participants were asked to verbalize their thinking while solving the problems. Participants were allowed to ask for help when they were stuck. After the practice problems, we asked learners to rank the three practice tools: micro Parsons problems with block-based feedback (PB), micro Parsons problems with execution feedback (PE), and text-entry questions (with execution-based feedback) (TE). Finally, we conducted a short semi-structured interview with each participant, with starter questions such as "What were you considering when you were ranking the problem types".

4.2 Results

One researcher first went through the transcriptions of the first four participants and used open coding with the grounded theory methodology [11] to generate an initial codebook with three code families. Following the initial code book, the researcher and a graduate student coded the transcript of four participants, while iterating on the code book. After two rounds of discussion and iteration, the two coders coded eight participants independently according to the revised code book, and reached a Krippendorf's alpha of 0.92. The researcher then coded the remaining four participants independently.

4.2.1 Learners' Preference for Practice Type, Self-Efficacy, and Familiarity with the Topic. Between the think-aloud practice and the interview, we asked learners to rank the types: Parsons with block-based feedback (PB), Parsons with execution-based feedback (PE), and text-entry (TE) (with execution-based feedback). Five participants (42%) picked TE as their most preferred type of problem for practice, four (33%) chose PE, and three (25%) chose PB. Table 2 shows the complete results for learners' preference rankings.

Table 2: Learners' Self-Reported Preference Ranking for Three Type of Practice Problems

Ranking	Count	Participants
PE > TE > PB	3	P01 P06 P09
TE > PE > PB	3	P02 P04 P05
PB > TE > PE	2	P03 P08
TE > PB > PE	2	P10 P12
PB > PE > TE	1	P11
PE > PB > TE	1	P07

For the presurvey on participants' self-efficacy in computing and familiarity with the topic, we calculated the overall score for each participant as the mean of all survey questions (6 questions for self-efficacy, 3 questions for familiarity with the topic), on a scale of 1 (low self-efficacy or familiarity) to 5 (high self-efficacy or familiarity). The average of participants' self-efficacy ratings was 3.79 (n = 12, SD = 0.33). Participants whose most preferred question type was text entry (n = 5) had an average self-efficacy in computing of 3.63 (SD = 0.30), and participants whose most preferred question type was Parsons (PE or PB, n = 7) had an average self-efficacy of 3.88 (SD = 0.62). In terms of familiarity with the three types of statements in the study (SELECT, UPDATE, and SELECT with JOIN), participants had an average score of 2.89 (SD = 0.96). Learners who picked text entry as their first choice had an average of 3.27 (SD = 0.83), and those who picked Parsons (PE or PB) had a relatively low average of 2.62 (SD = 1.00). Figure 5 shows the box plot of the results.

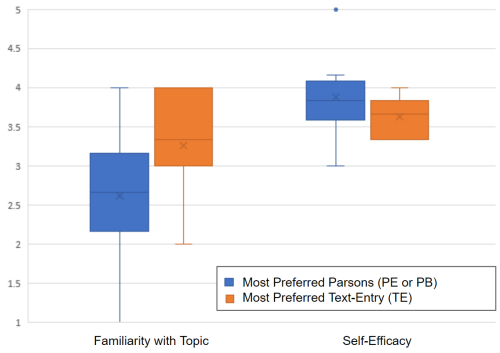


Figure 5: Box plot of self-reported self-efficacy and familiarity with the topic by the input type of the most preferred problem type.

4.2.2 Learners' Criteria for Preference. Right after the learners ranked the problem types, we asked for the underlying criteria that determined their preference, i.e. *"What were you considering*

when you were ranking them". We extracted three themes from the learners' responses.

Perceived learning. Although participants had different preferences for the tools, all participants said that they preferred tools that they perceived as helpful for their learning. However, learners had quite different ideas of what helps them learn best. Most participants compared the difference of perceived learning from the perspective input types (Parsons or text-entry).

For those who favor text entry, some reported that simply typing things out makes them feel like they are learning more, and some preferred to minimize the help they receive during practice:

> *"So it's like, just doing it yourself is the best way to learn it, because the more times you do it, the better you get at it..., obviously, it's [a] bit harder, because you aren't getting as much help. But I think that, like, that, fundamentally, is the best way to learn how to code."* (P05)

On the contrary, some learners who preferred Parsons (PE or PB) brought up the importance of learning efficiency, and getting help is an important part of that.

> *"My primary criteria was the one that was easiest for me to like, get the concept if I didn't already understand it... I don't think I would have been able to figure out just the text[-entry] 'JOIN' problem by myself."* (P11)

Meanwhile, some learners value getting the right type of feedback the most. P06 loved getting execution-based feedback, as it helped them feel more active in the learning process:

> *"So I know, I'm kind of more active in it. And I don't feel like I'm just kind of breezing through or zoning out by any means."* (P06)

Task Authenticity. In their responses, five participants (42%) mentioned that they value the authenticity of the task, four of which (80%) picked text-entry as their most preferred practice tool. The other participant picked PE as their first choice because it provided the same feedback as the text-entry problem. The learners expressed that they want to gain experience in an environment that is similar to how programming is used in life:

> *I mean, for practice's sake, I probably would rank this one [text-entry] as the first one. Because in real life, you still have to [do text] entry. Like you enter your code in text.* (P10)

More specifically, the authenticity aligns with the scenarios in which they will be tested in the future, such as technical interviews, and this consideration outweighs the perceived benefits of Parsons problems. For example, P12 felt that Parsons problems help learners focus on the big picture instead of spelling, but concluded with:

> "*But, you know, overall, I still prefer this [text-entry] because it just lets [me] challenge myself and, you know, test my skills as if it were an interview or something.*" (P12)

Prior Knowledge. Three participants (25%) explicitly expressed that their prior knowledge, or their current proficiency in the topic, played an important role when they decided their preference. All three of them thought that text-entry requires more prior knowledge to successfully complete. P01 described that evaluating their own level of proficiency is also part of their thinking process:

> "*So like, the text entry problems were where I was at, like, knowledge-wise. I think you definitely need to know more to do the text entry problems.*"

Two of these participants also explained the role of prior knowledge from the perspective of feedback when comparing the two types of Parsons problems. They felt that the block-based feedback was great as an introduction to knowledge, since highlighting the incorrect blocks gives them more direction on how to fix the problem:

> *"So since the block-based feedback, will, like visually tell you, Hey, this is where you got things wrong, I think these can be very helpful for practicing something they just got introduced to."* (P08)

4.2.3 How does the type of input affect the learning experience? Based on learners' feedback, we summarized the unique features of the learning experience using text-entry input and Parsons input. Text-entry input provides ***less help***, requires learners to ***pay more attention to details*** such as spelling and table/column names, and maintains the authentic ***typing experience***.

According to learners, when using text-entry, they have a lot more to consider than just moving the blocks to the correct place. Some participants valued the process of making and fixing small mistakes, as they feel that it is part of the learning process. When solving one of the text-entry problems, P01 accidentally added "1" instead of "10" as required, and fixed the mistake afterward. When reflecting on that, they commented:

> "*I've always been taught, like, the best way to learn is from making mistakes... it's definitely a lot harder to make mistakes on drag-and-drop problems. So I was able to learn and I think it was helpful being able to like, look through and figure out what I did wrong through the text entry problems.*" (P01)

However, some learners thought irrelevant mistakes were not helpful for focusing on what they were really trying to learn. P10, who made a similar mistake by writing "90" instead of "10" and spent a long time debugging it, said

> *"In that sense, I feel like drag and drop is more efficient, because changing this from 10 to 90 doesn't really, like, improve my skill. And SQL is my [current] problem."* (P10)

Interestingly, we found that the simple activity of typing can affect learners' experience. P02 compared the benefit they felt from text-entry input with the experience of note-taking:

> "*You know, it's kind of like when you write out notes in class or something, kind of just feel like you're like taking in the information a little bit better. At least I feel that way.*"

Compared with text-entry input, learners found Parsons ***faster*** to complete and ***more efficient for learning*** since they avoided making trivial mistakes. Learners also felt that Parsons problems provided them with ***worked examples for learning***. P07 compared

their perceptions of Parsons problems and text-entry problems' roles:

> "*I think the drag and drop ones are more like, an input process. And text-entry is more like, testing my knowledge after I have learned something.*"

However, learners also expressed concern that there's the possibility of ***gaming the system***, and worry some users might just reorder the blocks until the answer is correct, without learning from it.

4.2.4 How does the type of feedback in Parsons problems affect the learning experience? All participants felt that block-based feedback was easier to understand than execution-based feedback. We also summarized the unique traits of block-based feedback (fig. 3) and execution-based feedback (fig. 4) suggested by the learners.

According to the learners, block-based feedback is helpful in the way that it ***helps locate errors*** and ***uses natural language feedback***. One of the common features mentioned by the participants was the ability to highlight incorrect blocks (blocks that are not in the longest common sequence between the submitted answer and the correct answer). Some felt that it helped them to fix their solutions faster, and others felt it also helped them learn:

> "*If I have a block that's highlighted, and tells me that this isn't the wrong place. I think I learned better, and I will not make a similar mistake in the future.*" (P12)

Although when learners' answers are incorrect, block-based feedback always provides the same feedback text as shown in fig. 3, learners feel that the prompt, written in natural language, also helped them fix their code. P08 said:

> *"[after seeing the feedback,] and then you're like, which one was, say, the wrong order, or it was just the wrong option in general. And so I could kind of correct myself quickly."*

However, some learners also feel block-based feedback was ***not directly related to coding***, so they did not know why it was incorrect in the sense of programming.

On the other hand, execution-based feedback ***provides the original compiler error message*** and maintains ***the authenticity of debugging***. When a syntax error occurs in submitted code, execution-based feedback provides the error message as-is, which does not always point to the exact error location. P02 was frustrated by the error message:

> *"I can't really make sense of this... I also don't know the 'errors near FROM: syntax error'. But like, in my head, I kind of know that the issue is 'FROM' should come before 'WHERE', but this error is not really super helpful..."*

However, some participants like to figure out how to fix the errors, since they find that more challenging.

> "*I would probably prefer problems like this (execution-based) where I would want a little bit of challenge, so I can figure out how to how to solve those syntax errors on my own.*" (P03)

Participants also highlighted that execution-based feedback feels more authentic (see section 4.2.2).

5 BETWEEN-SUBJECTS FIELD STUDY

To compare the effectiveness of using micro Parsons problems as practice in SQL versus traditional text-entry problems, we conducted two between-subjects field studies in an authentic classroom environment to evaluate the differences in learners' learning gain and ability to adopt certain syntax patterns in SQL. Because the class we have access to has a limited class size, we split the study into two parts which were conducted in two consecutive semesters: First, in fall 2022 (semester 1), we compared the effectiveness of Parsons problem with block-based feedback and traditional text-entry problems in class (PB-TE1); Then, in winter 2023 (semester 2), we compared Parsons problems with execution-based feedback and traditional text-entry problems, in the same class (PE-TE2). Both classes was taught by the same instructor and had the same syllabus. The study was conducted in a data-oriented programming course. It used Python as the main programming language and included a variety of topics, including basic Python data structures, object-oriented programming, web scraping, etc. The course covered the topic of working with databases in three lectures, during which basic SQL concepts and skills were introduced, such as using CREATE TABLE, INSERT, SELECT, UPDATE, DELETE, and basic JOIN statements. The study was conducted in an R1 research university with IRB approval, and all data collected were anonymized. One of the authors is the instructor of the course. The instructor uses an interactive ebook in Runestone, which have traditional Parsons problems as practice. Students have practiced with traditional Parsons problems as homework and in-lecture activities regularly prior to being introduced to micro Parsons problems with SQL.

5.1 Methods

5.1.1 Participants and Procedure. To avoid taking too much time from the normal teaching activities in lectures, in each semester, we conducted the study in two sessions during the two consecutive lectures of the course. Fig. 6 demonstrates the study procedure for PB-TE1 (semester 1), the number of students who participated in each stage, and the final participant number. PE-TE2 adopted the same procedure, only the number of participants was different.

During the first lecture, the instructor briefly introduced the basic concepts of databases and tables, simple SELECT, UPDATE, INSERT, DELETE statements, and WHERE clauses with logical operators. At the end of the first lecture, students took a 10-minute timed pretest with three traditional text-entry problems. No assignments on databases or SQL were issued between the two lectures so that any learning gain from the pretest to posttest likely resulted from the practice activities in the study. The second part was conducted at the beginning of the second lecture, which happened two days after the first lecture. It started with an 8-minute knowledge introduction including three worked examples for SELECT, UPDATE, and JOIN. It reviewed the knowledge covered during the first lecture and introduced new knowledge prior to asking learners to practice. New concepts and syntax patterns introduced on the knowledge introduction page included using UPDATE to add to an existing value (e.g. "SET price = price + 2") and simple JOIN statements. Next, students moved on to a 15-minute practice, where they were randomly assigned to either the micro Parsons (Block-based feedback) group (PB) or the traditional text-entry group (TE) to

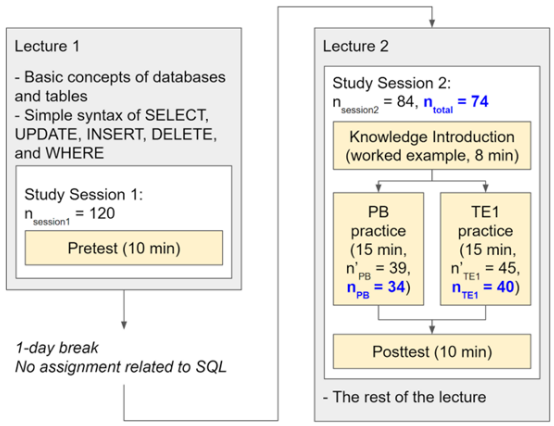


Figure 6: Procedure of study PB-TE1 in the first semesters in the context of the lectures. The number of students who showed up in the current session is denoted by black text, while the final participants who showed up for both lectures were highlighted in blue.

complete five practice problems. Finally, the second session ended with a 10-minute timed posttest, which contained three traditional text-entry problems that were isomorphic to the pretest. Learners were instructed not to refer to outside resources during the study. Students who showed up to both lectures and completed all activities in the study session were considered valid participants. In total, 74 students were included in the PB-TE1 study (semester 1, $n_{PB} = 34$, $n_{TE1} = 40$), and 46 students were included in the PE-TE2 study (semester 2, $n_{PE} = 16$, $n_{TE2} = 30$).

5.1.2 Study Material. The practice problems of study two used the last five practice problems from study one. The first problem from study one was not included in this study, as it was similar to the second problem.

Both the pretest and the posttest were a set of three text-entry problems with SELECT, UPDATE, and SELECT/JOIN statements to test learners' abilities to write code in an authentic environment. The pretest and posttest questions were isomorphic, meaning that the problems, solutions, and databases had the same structure, but in different contexts and had different data. As learners were new to SQL and were expected to produce many syntactically incorrect answers, automatically grading SQL statements based on execution would result in a lot of zero scores, leading to the loss of details that reflect students' understanding of certain keywords and patterns. Thus, we developed a grading scheme based on students' abilities to write out certain parts of the answer. A sample grading rubric for problem two in the pretest is presented in table 3.

After two researchers discussed and iterated on the rubrics, one researcher then manually graded students' answers for the pretest and the posttest. The full mark for each test is 10 points. During grading, the researchers were blind to the experimental group assignment of the students.

5.2 Results

Using the same methods in PB-TE1 (semester 1) and PE-TE2 (semester 2), we evaluated the learning gain and students' abilities to

Table 3: Grading Rubric for Test Problem 2

Correct Answer:
UPDATE equipment SET quantity = quantity + 5
WHERE sport = "table_tennis"

No.	Pattern	Position	Pt.
1	UPDATE equipment SET	-	1
2	SET quantity = quantity + 5	after **1**	1
3	WHERE sport = "table_tennis"	after **1**, **2**	1
4	met **1-3** but has extra incorrect code	-	-1

reproduce certain programming patterns that were provided in a single block as a worked example.

5.2.1 Learning Gain. We first checked if there was any significant difference between the groups by the pretest score. In the PB-TE1 (semester 1) comparison, group PB had an average pretest score of 1.9 (n = 37, SD = 2.2), and group TE1 had an average pretest score of 2.3 (n = 45, SD = 2.4). In the PE-TE2 (semester 2) comparison, the average pretest score of PE was 5.0 (n = 16, SD = 3.4), and the average score of group TE2 was 5.4 (n = 30, SD = 3.4). We used Welch's t-test to analyze the difference between the pretest score of PB and TE1, as well as PE and TE2, as [7, 37] recommended it in social science analysis, especially for small samples. No statistically significant difference was found at $\alpha = 0.05$, meaning the groups in each semester were comparable. When comparing between two semesters, we found that the winter 22 semester (PE-TE2) had a significantly higher pretest score (mean = 5.26) than the first semester (PB-TE1, mean = 2.13), suggesting that the population for the two semesters was not comparable. Thus, we did not perform any comparisons across the semesters.

We obtained the students' learning gain by subtracting the learners' pretest score from the isomorphic posttest score.

Table 4 shows the result of the Welch's t-test on the learners' learning gain. The average learning gain for students in the block-based feedback micro Parsons problems group (PB) was 3.27 (n=37, SD=2.99), and the average learning gain for its paired control group (TE1) was 1.82 (n=45, SD=2.64). There was a significant difference in the overall learning gain between PB and TE1 ($p = 0.024$, Hedge's $g = 0.52$). For the comparison between execution-based feedback micro Parsons (PE) and its control group (TE2), we did not find a significant difference at $\alpha = 0.05$ level. The average learning gain for PE was 4.38 (n=16, SD=3.24), and the learning gain for TE2 was 2.87 (n=30, SD=3.61).

5.2.2 Learning Code Patterns. We also analyzed learners' ability to learn from the code patterns provided by a single block in micro Parsons problems. We identified two patterns, each provided within a single block, such that learners can learn the usage of the patterns by putting the block that contains the pattern in the right place. The first pattern ("quantity = quantity + 5") comes from the UPDATE statement, where learners were asked to add a certain value to an existing column. To only evaluate the pattern, we ignored the formatting, and allowed any numbers. The second pattern ("students.name") comes from SELECT + JOIN statement, where learners were asked to specify a column name in a table. As long as the learner successfully called one column by a complete

Table 4: T-Test for Student Learning Gain

Parsons Feedback	Micro Parsons			Text-Entry			t	p	g
	n	$\overline{x}$	σ	n	$\overline{x}$	σ			
Block (PB-TE1)	37	3.27	2.99	45	1.82	2.64	2.3	0.024*	0.52
Execution (PE-TE2)	16	4.38	3.24	30	2.87	3.61	1.4	0.158	0.43

*$p < .05$, **$p < .01$., ***$p < .001$

table and column name in their answer, we marked the pattern as implemented. We used regular expressions to match learners' answers in the posttest to identify the number of learners who correctly implemented the patterns, and assigned a score of zero (did not implement) or one (implemented) for each pattern.

Table 5 and table 6 demonstrate the results of students' learning of code patterns. In the PB-TE1 comparison, learners in the PB group demonstrated a significantly higher learning gain for the first pattern ($p = 0.015$, Hedge's $g = 0.58$) compared to TE1, but no significant difference was found for the second pattern. Interestingly, in the PE-TE2 comparison, learners in the PE group demonstrated a significantly higher learning gain than TE2 for the second pattern, and had a large effect size ($p = 0.002$, Hedge's $g = 0.89$). However, no significant difference was found for the first pattern between PE and TE2.

6 DISCUSSION

In this section, we discuss our results and summarize our findings to answer the research questions.

6.1 RQ1: Student preference, and their criteria for preference.

We investigated learners' preference for the three types of practice questions for SQL (micro Parsons problems with Block-based feedback (PB), micro Parsons problems with execution-based feedback (PE), and text-entry problems with execution-based feedback (TE)) through a within-subjects study. By asking learners to rank their preferences, we found that learners generally perceived all three types of problems as helpful for learning, but had very diverse personal preferences. In our within-subjects study, around half of the participants chose text-entry problems as their most preferred way of practice, and half of the participants chose micro Parsons problem; This aligns with previous findings on Parsons problems [13]. Most people find Parsons problems useful for learning to code, but some would rather write code than solve a Parsons problem, and some would like the choice to switch between Parsons problems and write-code problems. In regards to Parsons problems' feedback, learners also demonstrated diverse preferences: around half of the participants preferred execution-based feedback, while the other half preferred block-based feedback.

To understand learners' rationales for their preferences, we took a deeper dive and asked learners to explain the underlying criteria for their preferences. When asked about their preference, the learners' responses mostly focused on comparing micro Parsons problems and text-entry problems. We found that perceived learning, task authenticity (similarity to professional practices), and prior knowledge are the most important considerations. While all learners perceived learning as the most important criterion, they had very different opinions on what helps them learn best. In our think-aloud study, we found that the average level of familiarity with the SQL concepts of students who preferred text-entry was higher than the average of the students who preferred Parsons. This aligns with learners' explanations for their preferences. Many students who preferred text-entry problems highly valued being challenged, and viewed the practice as a "test" for themselves. Their goal was to accomplish tasks from scratch with the minimum amount of hints or scaffolding. In contrast, many learners who preferred micro Parsons problems viewed the practice as an "knowledge input" process. They valued the opportunity to gradually familiarize themselves with new concepts by actively interacting and experimenting with them. Meanwhile, we found that task authenticity, or how similar the process of solving a practice problem is to a real-world scenario that the students care about (e.g. tests, interviews, professional practice), is an important consideration, and was mentioned by most students who preferred text-entry. This suggests the design potential for customizing the presentation of Parsons problems for learners: for learners who strongly prefer the "real programming" experience, instead of making the drag-and-drop interface the only option, Parsons problems can be used as just-in-time scaffolding when requested by learners as hints, such as in [15]. Or, a mixed-modality input that enables both typing and using provided blocks can potentially help learners have a stronger sense of authenticity while giving them agency over their learning.

6.2 RQ2: Perceived Advantages and Disadvantages for Input Type

When asked about input types and feedback types, learners identified some common traits but demonstrated different opinions in terms of perceiving them as advantages or disadvantages.

Some students preferred text-entry problems, and viewed the more complicated nature of text-entry problems as an advantage. They liked to have the ability to make mistakes during practice questions, and spend time correcting their own mistakes with less help, viewing it as a valuable learning process. Naturally, they found micro Parsons problems provided too much help, and worried that getting the micro Parsons problem correct does not mean that they can write the equivalent code. In contrast, other students who viewed micro Parsons problems as a learning process viewed the reduced problem space as an advantage, because it helped them focus on the important concepts and made learning more efficient.

Table 5: T-Test for Pattern Acquisition: Block-Based Parsons Versus Text-Entry

Pattern	Parsons (n = 37)		Text-Entry (n = 45)		z	p	g
	n	%	n	%			
"quantity = quantity + 10"	0.54	0.60	0.24	0.43	2.49	0.015*	0.58
"table.column"	0.30	0.49	0.29	0.46	0.32	0.79	0.02

*$p < .05$, **$p < .01$., ***$p < .001$

Table 6: T-Test for Pattern Acquisition: Execution-Based Parsons Versus Text-Entry

Pattern	Parsons (n = 16)		Text-Entry (n = 30)		z	p	g
	n	%	n	%			
"quantity = quantity + 10"	0.50	0.52	0.30	0.60	1.18	0.244	0.35
"table.column"	0.88	0.34	0.47	0.51	3.24	0.002**	0.89

*$p < .05$, **$p < .01$., ***$p < .001$

For these learners, text-entry problems can create unnecessary frustration during learning.

Interestingly, different from Wu et al.'s work [40] of using micro Parsons problems to scaffold the learning of regular expressions (regex), none of our participants mentioned that they had more freedom in text-entry input or found that the micro Parsons condition was restrictive. This is potentially due to the difference between SQL and regex. SQL is more structured, and usually only has one correct solution for beginners' practices. Regex is more flexible and can have multiple solutions, which makes it frustrating when more advanced learners want to construct a solution that is different from the intended answer. Another reason could be that one of the important learning goals for SQL novices is to familiarize themselves with the syntax structure, while in regex there are no fixed structures to follow, only patterns to learn. This comparison also highlights the importance of testing teaching approaches in different context. The nature of different learning content, in our case, different programming languages, can result in different experiences for learners.

6.3 RQ3: Perceived Advantages and Disadvantages for Feedback Types of micro Parsons Problems

The recent literature review on Parsons problems pointed out that there's not enough research on different types of feedback [9]: execution-based versus block-based. To help fill that gap we asked learners to compare their experiences using the two types of feedback. Although learners also demonstrated diverse preferences towards the feedback, compared to the input types, they agreed more on some of the advantages and disadvantages of block- and execution-based feedback.

Execution-based feedback provides the original compiler message, and thus maintains the authenticity of debugging for learners. However, they lack instructions that guide learners towards the correct solution, do not tell them exactly what is wrong, and are more difficult, especially for beginners. This aligns with Helminen et al.'s [14] student survey findings. Meanwhile, block-based feedback uses natural language and is more friendly for novices. Highlighting the blocks give students a better sense of direction to fix their code, but the detachment from coding knowledge-related feedback is viewed as a disadvantage.

This suggests the potential of combining the advantages of block-based feedback and execution-based feedback, by developing a type of feedback that points learners to the direction of fixing the error more directly, and embeds learning-related information at the same time.

6.4 RQ4: Short-Term Learning Gain and Pattern Acquisition

In two separate semesters, we compared micro Parsons problems with block/execution-based feedback with the traditional type of practice of writing code from scratch.

We found that learners who used micro Parsons problems with block-based feedback had a significantly higher learning gain than the traditional text-entry group, and the execution-based feedback practices were equally effective as text-entry problems. This finding suggests that using micro Parsons problems as programming puzzles is effective for novices learning SQL.

It is interesting to note that PB-TE1 (semester 1) had a lower pretest score than semester 2, indicating their level of prior knowledge is relatively lower. Combined with our findings in the qualitative study, where participants who preferred micro Parsons problems had a lower average prior knowledge, it is possible that for students with lower prior knowledge, micro Parsons problems can result in a higher learning gain. However, the different types of feedback that were used between the two semesters, as well as the difference in population can also be factors affecting learning gain. Further research could be done to evaluate micro Parsons problems' effects on learners with different prior knowledge on the topic.

In terms of pattern acquisition, students in the two semesters demonstrated different changes. For the PB-TE1 group (semester 1),

learners in the micro Parsons with block-based feedback group had a significantly higher pattern acquisition than the text-entry group in the "quantity = quantity + 10" pattern, which was included as one block in the micro Parsons problem. Although the patterns were shown for both groups as worked examples prior to the practice session, the Parsons groups had higher pattern acquisition. This suggests the block could have been used as an interactive worked example, allowing learners to get familiar with how to use the pattern, without having to memorize all details. On the other hand, learners had comparable pattern acquisition on the "table.column" pattern. It is possible that the learners learned from similar patterns in object-oriented programming ("object.attribute"), which was covered before SQL in the course.

For the PB-TE2 group (semester 2), which has a higher pretest score, learners had comparable pattern acquisition on the first pattern, but the PE group had significantly higher acquisition on the second pattern. When we took a closer look at students' responses, we discovered that for the second pattern, many students with higher levels of prior knowledge used aliases to refer to table names in the pretest, and were not counted as the pattern as it was not demonstrated in Parsons problems. However, in posttest, some learners changed their answers to follow the blocks shown in the micro Parsons problems: using full table names instead of aliases. This could result in an increase in pattern acquisition for the Parsons group, as students recognized the pattern and decided to follow the example provided in the blocks.

It is important to note that while we provided statistic comparisons between two different groups in each semester, the comparison across semesters is only our interpretation of the result, as there are two different changes across the semesters: the feedback for Parsons problem, as well as student demographics. Future work is needed to investigate how these two factors influence the effectiveness of Parsons problems as practice.

6.5 Micro Parsons Problems and Novices' Misconceptions in SQL

In this setion, we connect our study and system with prior work in novices' misconceptions in SQL, and discuss the benefits of using micro Parsons problems to help learners avoid misconceptions.

Through a think-aloud study, Miedema et al. [19] explored the common reasons for novices' misconceptions in formulating SQL queries. The majority of the misconceptions they identified were closely related to SQL syntax. Novices often incorrectly transferred their knowledge in other programming languages, math, or natural language to SQL. As a result, they confuse the use of keywords and symbols in SQL, such as "==" and "=", whereas in other programming languages, the former ("==") is often used for value comparison, and the latter ("=") is used for value assignment.

In other words, syntax is not a trivial issue for SQL novices. In our think-aloud study, we also observed that when learners were facing text-entry problems, they tended to struggle with the correct syntax. Micro Parsons problems, however, enabled learners to explore their answers with a given set of keywords, and reduce the burden of memorizing the exact keyword. Micro Parsons problems were also helpful for pointing out the common confusion of symbols by including distractors.

Even with generative AI to help users fix syntax errors, we believe practicing and understanding syntax in the case of SQL is important. As pointed out by Miedema et al. [19], the syntax issue in SQL for novices is not simply memorizing how to use them, but the confusion caused by the "(in)consistency" of the language. For example, in SQL, when defining aliases for tables in SELECT statements, the alias can be used before its definition ("SELECT p.name FROM product AS p", where the "AS" keyword is optional). At the same time, aliases can also be created for attributes ("SELECT name AS n FROM product", where the "AS" keyword is also optional). This inconsistency is confusing for learners, leading them to make mistakes such as "SELECT name p FROM product p"[2]. Automatically fixing this error does not reduce the confusion for learners. Micro Parsons problems with distractors can help learners directly contrast the incorrect and correct usages and highlight their differences.

For logical and semantic misconceptions, prior work also found that novices tend to incorrectly generalize SQL templates to other problems. For these types of misconceptions, micro Parsons problems can be used to contrast correct and incorrect use of templates for a given problem, as demonstrated in fig. 7.

Figure 7: An example of a more complicated SQL problem using COUNT and GROUP BY, from the example of using incorrect templates by [19]. The correct answer should be "SELECT city," "COUNT(sID)" "FROM" "store" "GROUP BY" "city". The block "COUNT(city)" is a distractor, as learners who memorize the "COUNT - GROUP BY" template can still confuse the correct column to count.

6.6 Reflecting on Micro Parsons Problems from a Design Perspective

In this section, we reflect on the design of micro Parsons problems by comparing them with traditional Parsons problems, and propose potential designs that could resonate with learners' need for authenticity.

6.6.1 Micro Parsons problems versus traditional Parsons problems: more than just scale differences. The primary motivation for using micro Parsons problems instead of traditional Parsons problems for SQL was that SQL practice is often limited to one statement. Several findings for our study also echoed the prior work on traditional Parsons problems, such as learners have mixed preferences towards Parsons problems and writing code, and perceive Parsons problems

[2]Example are simplified student errors by Miedema et al. [19]

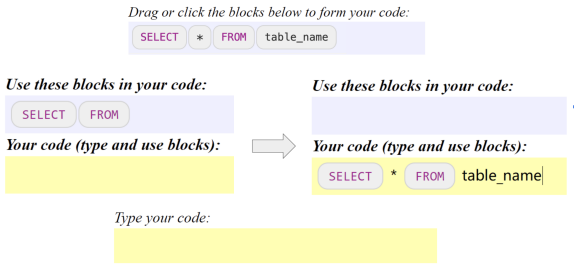


Figure 8: An example of using input modality in micro Parsons problems to adjust the difficulty, ranging from providing all blocks, asking learners to use a mix of blocks and typing, and typing from scratch.

as easier. However, we also discovered some unique potential for micro Parsons problems.

For example, during the interview, all learners agreed that micro Parsons problems are "a different type of input", rather than a completely different type of problem. As a one-statement problem, micro Parsons problems' input area is limited to one line. The interface allows users to either click on a block to add it to the end of their statements, or drag and drop a block to a desired position. During the think-aloud study, most participants primarily clicked blocks to create their statement from left to right, closely resembling writing a statement from scratch. Drag-and-drop methods were mostly used to rearrange blocks users already put in their statements. This has not been reported by prior research on traditional Parsons problems.

Another difference of micro Parsons problems is the flexibility of deciding the size of code blocks. In previous implementations of traditional Parsons problems, some only allow one line in each block, such as Epplets [18]. For those that allow one or more lines in each block, such as Runestone [21] and PrairieLearn [36], the largest block usually does not exceed three lines. We did not find existing research studying the block size for traditional Parsons problems. The reason for avoiding larger blocks could be to prevent the large blocks from being too difficult to comprehend. In contrast, because of the small size of micro Parsons problems, we can easily have some blocks with four or more words/symbols (e.g. "grade = grade + 10") without being a burden to learners. Thus, micro Parsons problems could be more flexible in introducing larger blocks to present a complete code pattern.

6.6.2 Improving programming learning tools for more "authentic" practice. During our think-aloud study, we observed that a lot of learners prefer high authenticity in the practice environment. They expect the practice environment to be very similar to the application scenarios that are meaningful for them, such as exams and interviews. This could be because when practicing in environments with high authenticity, learners are more confident about their self-evaluation. To help fulfill learners' needs for more authentic practice while providing scaffolding, we can improve the design in two ways. First, increase the resemblance of the interaction method between the practice environment and the authentic task; Second, provide transitions from the practice environment to the authentic task to help learners track their progress toward the authentic task.

Based on these two methods, we propose a future design for micro Parsons problems that use a mixed input modality of blocks and text-entry. t is more similar to the text-entry task than only using code blocks, and makes learners aware of how much help they are getting from the practice environment.

7 LIMITATIONS AND FUTURE WORK

7.1 Limitations

One limitation of our work involving block-based feedback is the choice of algorithm for highlighting incorrect blocks. In this implementation, we chose LCS to stay consistent with the design of traditional Parsons problems in the same platform, and to avoid confusing learners who have used the traditional Parsons problems in the same platform before. However, the choice of algorithm can also affect learners' perceptions of the block-based feedback mechanism, and our results are limited to block-based feedback that adopt the LCS algorithm.

For our qualitative study, one of the major limitations is volunteer bias. While we discovered that different levels of prior knowledge affected their preferences, we also acknowledge that the participants in the qualitative study might not accurately represent the population of novice learners. The think-aloud participants could be more confident in programming, more knowledgeable in SQL, or more open to trying out new tools for learning.

Our quantitative study was also limited in several ways. First, our study was conducted in two different lectures, meaning that only learners who came to both lectures and completed all activities participated in our study, resulting in a potential selection bias. Next, although we made sure there were no related practices or homework between the pretest and the rest of the study, learners could still learn during that period. Finally, as we have discussed in section 6.4, we were unable to gather enough evidence to compare the effectiveness between feedback types, as students in the two semesters demonstrated significantly different level of prior knowledge.

7.2 Future Work

First, we discuss several future directions for Parsons problems research. Based on our findings and limitations, there is potential to investigate two factors on the effectiveness of micro Parsons problems for learning. On the students' side, how does prior knowledge affect the effectiveness? Does syntax knowledge matter more than semantic and logic knowledge? On the design of micro Parsons problems, there is also more research to be done to explore different feedback's effects on student learning.

In terms of learning, this work focused on providing a comparison of short-term learning and pattern acquisition for SQL beginners who just started to learn a new concept, using micro Parsons problems and traditional text-entry problems. Although our study design avoided having other instructional activities assigned that might affect the results, we did not include a delayed posttest to evaluate knowledge retention. Future work should explore the long-term effect of practicing with micro Parsons problems versus text-entry problems in SQL. Our work in this study also primarily focused on beginner-level SQL, and did not go into more advanced

content. We intend to examine the tools' ability to assist learners in more advanced SQL content, such as complicated clauses.

With regard to the context of micro Parsons problems, we have found that learners had different feedback for micro Parsons in SQL and regex. Future work could expand to different contexts, such as using one line of micro Parsons problems within a larger code piece, exploring its effect on other languages, etc. Future work can also compare how learners' perceptions of micro Parsons problems differ from Parsons problems, in terms of authenticity and effects on their self-efficacy.

Second, for programming puzzles and learning puzzles in general, our results pointed out two important considerations for design: task authenticity and learner agency. As discussed in section 4.2.2, students had very diverse personal preferences; Some students had very strong preferences towards tasks that were meaningful to them. Future work can also expand our findings to design learning tools that provide students with a sense of authenticity, such as pointing out the connection between the puzzle and knowledge, providing mixed modality such that the puzzle is connected to the authentic task, and enabling the transition between the puzzle and real-world practice.

8 CONCLUSION

We investigated the potential of using a type of programming puzzle, micro Parsons problems, to help students learn SQL. We implemented a system that provides SQL puzzles with different types of feedback. With a think-aloud study, we investigated learners' preferences towards different types of practice problems and the underlying reasons for learners' preferences. With two between-subjects classroom studies, we discovered that learners who used SQL micro Parsons puzzles with block-based feedback to practice had a significantly higher learning gain than traditional text-entry problems, and SQL micro Parsons problems with execution-based feedback are equally effective as text-entry problems. Our work provides a new way of practicing SQL and demonstrates the potential of using micro Parsons problems to help students with new concepts, especially those with less prior knowledge. Based on our studies, we also suggest changes to programming puzzles to provide options to support task authenticity and learners' agency.

ACKNOWLEDGMENTS

We thank all participants who volunteered to be part of the study. We also thank Xingjian Gu for helping with the qualitative coding of the think-aloud data.

REFERENCES

[1] Alberto Abello, Xavier Burgues, M. Jose Casany, Carme Martin, Carme Quer, M. Elena Rodriguez, Oscar Romero, and Toni Urpi. 2016. A Software Tool for E-Assessment of Relational Database Skills. *INTERNATIONAL JOURNAL OF ENGINEERING EDUCATION* 32, 3, A, SI (2016), 1289–1312. 20th Annual Conference on Innovation and Technology in Computer Science Education, Vilnius, LITHUANIA, JUL 06-08, 2015.

[2] Aisha Al-Salmi. 2018. A Web-based Semi-Automatic Assessment Tool for Formulating Basic SQL Statements: Point-and-Click Interaction Method. *Proceedings of the 10th International Conference on Computer Supported Education* 1 (2018), 191–198. https://doi.org/10.5220/0006671501910198

[3] Mark Ardis, David Budgen, Gregory W Hislop, Jeff Offutt, Mark Sebern, and Willem Visser. 2015. SE 2014: Curriculum guidelines for undergraduate degree programs in software engineering. *Computer* 48, 11 (2015), 106–109.

[4] Berkeley. 2023. Snap! Build Your Own Blocks — snap.berkeley.edu. https://snap.berkeley.edu/. [Accessed 10-12-2023].

[5] Peter Brusilovsky, Sergey Sosnovsky, Michael V Yudelson, Danielle H Lee, Vladimir Zadorozhny, and Xin Zhou. 2010. Learning SQL programming with interactive tools: From integration to personalization. *ACM Transactions on Computing Education (TOCE)* 9, 4 (2010), 1–15.

[6] Mark Chatham. 2012. *Structured Query Language By Example-Volume I: Data Query Language*. Lulu. com, Raleigh, NC.

[7] M. Delacre, D. Lakens, and C. Leys. 2017. Why psychologists should by default use Welch's t-test instead of student's t-test. *International Review of Social Psychology* 30, 1 (5 April 2017), 92–101. https://doi.org/10.5334/irsp.82

[8] Paul DiMaggio. 1997. Culture and Cognition. *Annual Review of Sociology* 23, 1 (1997), 263–287. https://doi.org/10.1146/annurev.soc.23.1.263 arXiv:https://doi.org/10.1146/annurev.soc.23.1.263

[9] Barbara J. Ericson, Paul Denny, James Prather, Rodrigo Duran, Arto Hellas, Juho Leinonen, Craig S. Miller, Briana B. Morrison, Janice L. Pearce, and Susan H. Rodger. 2022. Parsons Problems and Beyond: Systematic Literature Review and Empirical Study Designs. In *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education* (<conf-loc>, <city>Dublin</city>, <country>Ireland</country>, </conf-loc>) *(ITiCSE-WGR '22)*. Association for Computing Machinery, New York, NY, USA, 191–234. https://doi.org/10.1145/3571785.3574127

[10] Barbara J. Ericson, James D. Foley, and Jochen Rick. 2018. Evaluating the Efficiency and Effectiveness of Adaptive Parsons Problems. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (Espoo, Finland) *(ICER '18)*. Association for Computing Machinery, New York, NY, USA, 60–68. https://doi.org/10.1145/3230977.3231000

[11] Barney Glaser and Anselm Strauss. 1999. *Discovery of Grounded Theory: Strategies for Qualitative Research*. Routledge, New York. https://doi.org/10.4324/9780203793206

[12] Ryan Hardt and Esther Gutzmer. 2017. Database Query Analyzer (DBQA): A Data-Oriented SQL Clause Visualization Tool. In *Proceedings of the 18th Annual Conference on Information Technology Education* (Rochester, New York, USA) *(SIGITE '17)*. Association for Computing Machinery, New York, NY, USA, 147–152. https://doi.org/10.1145/3125659.3125688

[13] Carl C. Haynes and Barbara J. Ericson. 2021. Problem-Solving Efficiency and Cognitive Load for Adaptive Parsons Problems vs. Writing the Equivalent Code. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (<conf-loc>, <city>Yokohama</city>, <country>Japan</country>, </conf-loc>) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 60, 15 pages. https://doi.org/10.1145/3411764.3445292

[14] Juha Helminen, Petri Ihantola, Ville Karavirta, and Satu Alaoutinen. 2013. How Do Students Solve Parsons Programming Problems? – Execution-Based vs. Line-Based Feedback. In *2013 Learning and Teaching in Computing and Engineering*. IEEE, New York, NY, USA, 55–61. https://doi.org/10.1109/LaTiCE.2013.26

[15] Xinying Hou, Barbara Jane Ericson, and Xu Wang. 2022. Using Adaptive Parsons Problems to Scaffold Write-Code Problems. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1* (Lugano and Virtual Event, Switzerland) *(ICER '22)*. Association for Computing Machinery, New York, NY, USA, 15–26. https://doi.org/10.1145/3501385.3543977

[16] Sven Jacobs and Steffen Jaschke. 2021. SQheLper: A block-based syntax support for SQL. In *2021 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, New York, NY, USA, 478–481. https://doi.org/10.1109/EDUCON46332.2021.9453897

[17] Carsten Kleiner, Christopher Tebbe, and Felix Heine. 2013. Automated grading and tutoring of SQL statements to improve student learning. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '13)*. Association for Computing Machinery, New York, NY, USA, 161–168. https://doi.org/10.1145/2526968.2526986

[18] Amruth N. Kumar. 2018. Epplets: A Tool for Solving Parsons Puzzles. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 527–532. https://doi.org/10.1145/3159450.3159576

[19] Daphne Miedema, Efthimia Aivaloglou, and George Fletcher. 2022. Identifying SQL Misconceptions of Novices: Findings from a Think-Aloud Study. *ACM Inroads* 13, 1 (feb 2022), 52–65. https://doi.org/10.1145/3514214

[20] Daphne Miedema and George Fletcher. 2021. SQLVis: Visual Query Representations for Supporting SQL Learners. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, New York, NY, USA, 1–9. https://doi.org/10.1109/VL/HCC51201.2021.9576431

[21] Bradley N. Miller and David L. Ranum. 2012. Beyond PDF and ePub: Toward an Interactive Textbook. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (Haifa, Israel) *(ITiCSE '12)*. ACM, New York, NY, USA, 150–155. https://doi.org/10.1145/2325296.2325335

[22] MIT. 2023. Scratch - Imagine, Program, Share — scratch.mit.edu. ttps://scratch.mit.edu/. [Accessed 10-12-2023].

[23] Ioanna Moraiti, Anestis Fotoglou, and Athanasios Drigas. 2022. Coding with Block Programming Languages in Educational Robotics and Mobiles, Improve Problem

Solving, Creativity & Critical Thinking Skills. *International Journal of Interactive Mobile Technologies* 16, 20 (2022), 59–78. https://doi.org/10.3991/ijim.v16i20.34247

[24] Joint Task Force on Computing Curricula. 2013. *Computer Science Curricula 2013*. Association for Computing Machinery, New York, NY, USA.

[25] Dale Parsons and Patricia Haden. 2006. Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52* (Hobart, Australia) *(ACE '06)*. Australian Computer Society, Inc., AUS, 157–163.

[26] Nicolai Pöhner, Timo Schmidt, André Greubel, Martin Hennecke, and Matthias Ehmann. 2019. BlocklySQL: A New Block-Based Editor for SQL. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education* (Glasgow, Scotland, Uk) *(WiPSCE '19)*. Association for Computing Machinery, New York, NY, USA, Article 4, 2 pages. https://doi.org/10.1145/3361721.3362104

[27] Kai Presler-Marshall, Sarah Heckman, and Kathryn Stolee. 2021. SQLRepair: Identifying and repairing mistakes in student-authored SQL queries. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, IEEE, New York, NY, USA, 199–210.

[28] Shazia Sadiq, Maria Orlowska, Wasim Sadiq, and Joe Lin. 2004. SQLator: an online SQL learning workbench. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Leeds, United Kingdom) *(ITiCSE '04)*. Association for Computing Machinery, New York, NY, USA, 223–227. https://doi.org/10.1145/1007996.1008055

[29] David H. Smith, Max Fowler, and Craig Zilles. 2023. Investigating the Role and Impact of Distractors on Parsons Problems in CS1 Assessments. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (<conf-loc>, <city>Turku</city>, <country>Finland</country>, </conf-loc>) *(ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 417–423. https://doi.org/10.1145/3587102.3588819

[30] SQLsnap. 2023. SQLsnap! snap! with some extensions — snapextensions.uni-goettingen.de. https://snapextensions.uni-goettingen.de/. [Accessed 10-12-2023].

[31] John Sweller. 2006. The worked example effect and human cognition. *Learning and Instruction* 16, 2 (2006), 165–169. https://doi.org/10.1016/j.learninstruc.2006.02.005 Recent Worked Examples Research: Managing Cognitive Load to Foster Learning and Transfer.

[32] John Sweller, Jeroen JG Van Merrienboer, and Fred GWC Paas. 1998. Cognitive architecture and instructional design. *Educational psychology review* 10 (1998), 251–296.

[33] Toni Taipalus and Ville Seppänen. 2020. SQL education: A systematic mapping study and future research agenda. *ACM Transactions on Computing Education (TOCE)* 20, 3 (2020), 1–33.

[34] Jeroen JG Van Merrienboer and John Sweller. 2005. Cognitive load theory and complex learning: Recent developments and future directions. *Educational psychology review* 17 (2005), 147–177.

[35] Nathaniel Weinman, Armando Fox, and Marti A. Hearst. 2021. Improving Instruction of Programming Patterns with Faded Parsons Problems. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (<conf-loc>, <city>Yokohama</city>, <country>Japan</country>, </conf-loc>) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 53, 4 pages. https://doi.org/10.1145/3411764.3445228

[36] Matthew West, Geoffrey L. Herman, and Craig Zilles. 2015. PrairieLearn: Mastery-based Online Problem Solving with Adaptive Scoring and Recommendations Driven by Machine Learning. In *2015 ASEE Annual Conference & Exposition*. ASEE Conferences, Seattle, Washington, 26.1238.1 – 26.1238.14. https://doi.org/10.18260/p.24575 https://peer.asee.org/24575.

[37] Robert M West. 2021. Best practice in statistics: Use the Welch t-test when testing the difference between two groups. *Annals of Clinical Biochemistry* 58, 4 (2021), 267–269.

[38] Eric Wiebe, Laurie Ann Williams, Kai Yang, and Carol S Miller. 2003. *Computer science attitude survey*. Technical Report. North Carolina State University. Dept. of Computer Science.

[39] Joseph B Wiggins, Joseph F Grafsgaard, Kristy Elizabeth Boyer, Eric N Wiebe, and James C Lester. 2017. Do you think you can? the influence of student self-efficacy on the effectiveness of tutorial dialogue for computer science. *International Journal of Artificial Intelligence in Education* 27, 1 (2017), 130–153.

[40] Zihan Wu, Barbara J. Ericson, and Christopher Brooks. 2023. Using Micro Parsons Problems to Scaffold the Learning of Regular Expressions. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (<conf-loc>, <city>Turku</city>, <country>Finland</country>, </conf-loc>) *(ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 457–463. https://doi.org/10.1145/3587102.3588853

[41] Rui Zhi, Min Chi, Tiffany Barnes, and Thomas W. Price. 2019. Evaluating the Effectiveness of Parsons Problems for Block-based Programming. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19)*. Association for Computing Machinery, New York, NY, USA, 51–59. https://doi.org/10.1145/3291279.3339419