AI-Assisted Generation of SQL Comprehension Questions*

Martin Goodfellow[1], Alasdair Lambert[1], Andrew Fagan[1] and Robbie Booth[1]

[1]University of Strathclyde, Glasgow, Scotland, UK

Abstract

Students often struggle to fully comprehend the SQL queries they write. These gaps in understanding may remain hidden until later in their studies. At that stage, misconceptions about relational concepts, query semantics, formulation, or execution behaviour are difficult to remediate, particularly when students encounter increasingly complex queries. The increasing use of GenAI tools (e.g., GitHub Copilot) for query construction can further mask conceptual weaknesses.

A common instructional strategy for addressing this issue is the use of SQL comprehension questions that require students to predict query results, explain query logic, or identify semantic errors. Such questions help surface misconceptions and may also support academic integrity by requiring students to demonstrate conceptual understanding beyond producing a working query. However, creating scalable sets of high-quality questions is labour-intensive.

This tool demonstration presents an openly available extension of a prior GenAI-based system that automatically generates Java code comprehension questions, adapted to the domain of SQL. The tool automatically derives multiple-choice SQL comprehension questions directly from queries and integrates with the CodeRunner automated assessment platform to support rapid generation and deployment of scalable, personalised assessments. It is designed to support timely feedback, reduce instructor workload, and support conceptual understanding of SQL.

Keywords

SQL, comprehension, database, data systems, GPT, GPT-4o mini, generative artificial intelligence, GenAI, automated marking, CodeRunner

1. Introduction

Students in introductory database and data management courses frequently produce SQL queries that return the expected output, yet may not fully understand how those queries operate [1]. This pattern aligns with findings from introductory programming contexts, where students can write code that appears correct despite holding incomplete or inaccurate mental models of program behaviour [2]. Recent work in SQL education similarly reports that novices often struggle with core relational concepts and query semantics, leading to persistent misconceptions about how SQL evaluates queries [1, 3, 4]. SQL's declarative nature can make these misconceptions difficult to detect, as intermediate results are not visible and incorrect reasoning can produce apparently correct outputs on simple tasks.

Comprehension-oriented tasks have been shown to provide insight into how students interpret and reason about program behaviour [5]. When applied to SQL, comprehension questions offer instructors a way to probe students' understanding of query intent and expected behaviour, helping to surface conceptual gaps that may not be evident from the final output alone. However, generating tailored comprehension questions for each student submission is time-consuming and difficult to scale for large cohorts. Pre-prepared question banks may be perceived as less relevant to students' own work and are susceptible to reuse across cohorts [6, 7]. Automated feedback tools, including those used for large-scale practice and assessment, have shown promise for supporting scalable formative feedback [8, 9, 10], suggesting potential for similar approaches in SQL.

DataEd'26: 5th International Workshop on Data Systems Education

* Published in the Proceedings of the Workshops of the EDBT/ICDT 2026 Joint Conference (March 24-27, 2026), Tampere, Finland

martin.h.goodfellow@strath.ac.uk (M. Goodfellow); alasdair.lambert@strath.ac.uk (A. Lambert); andrew.fagan@strath.ac.uk (A. Fagan); robbie@tech.scot.ac.uk (R. Booth)

0000-0003-2151-8442 (M. Goodfellow); 0000-0002-9762-2193 (A. Lambert); 0000-0001-9714-2096 (A. Fagan); 0009-0002-6178-9300 (R. Booth)

To support personalised and scalable feedback in database courses, we present a tool that automatically generates multiple-choice SQL comprehension questions aligned with a student's submitted query. This tool extends prior GenAI-based question generation work [11, 12] by incorporating SQL-specific analysis. The goal of the system is to assist instructors in identifying misconceptions, provide timely formative feedback, and support the development of relational reasoning skills in large or frequently assessed cohorts.

2. Related Work

A substantial body of research shows that students frequently produce SQL queries that return the expected output while still holding incomplete or inaccurate understanding of relational concepts and query semantics. In a qualitative think-aloud study, Miedema et al. [1] identified a range of novice misconceptions, including misapplied prior knowledge, unstable mental models of relational operations, and misunderstandings of joins and grouping behaviour. These misconceptions often remain hidden during task completion, as students tend to judge their understanding based on whether a query executes successfully rather than on the soundness of the underlying reasoning.

Complementing these findings, a large-scale analysis of more than 8,000 student SQL queries showed that learners frequently overuse parentheses, introduce redundant joins, or construct unnecessarily complex solutions even when the final output is correct [3]. Such behaviour indicates that correctness alone is an unreliable indicator of comprehension. A Delphi study by Miedema et al. [4] further reports expert consensus that SQL query formulation poses persistent conceptual challenges for novices, who often make semantic and structural errors despite producing syntactically valid or executing queries. Additional evidence comes from SQLRepair [13], which demonstrates that student-written queries routinely contain logical or semantic issues that execution-based grading alone cannot detect.

More broadly, studies of SQL learning reveal consistent patterns of conceptual difficulty. Taipalus et al. [14] identify widespread semantic and logical errors in student-authored queries; Migler and Dekhtyar [15] trace student learning trajectories and document points of conceptual breakdown not immediately visible from correctness alone; and Shin [16] provides empirical evidence that the conceptual-to-SQL mapping involves substantial "semantic distance", helping to explain why students may generate syntactically correct statements without fully internalising relational semantics. Collectively, these findings underscore the need for assessments that probe students' reasoning rather than solely their ability to produce working queries.

Comprehension-oriented tasks, such as predicting a query's output or explaining its behaviour can reveal students' underlying reasoning and misconceptions [1]. However, such tasks are traditionally authored manually and do not scale to provide personalised feedback for individual submissions. Moreover, correctness-focused assessment tools often fail to expose deeper conceptual issues, as students' semantic and logical challenges are not reliably identifiable from final output alone [14, 15, 13]. Unlike corrective feedback, comprehension questions explicitly require students to articulate or select reasoning, making misconceptions observable rather than merely correctable.

Recent advances in GenAI have enabled the automatic generation of code-based comprehension questions. AutoMCQ [11, 12] demonstrates that multiple-choice comprehension questions can be generated directly from student-written code, providing scalable assessment of conceptual understanding beyond correctness. Complementary GenAI-based approaches have been explored in programming education, but in different assessment forms, including the use of Large Language Models (LLMs) to automatically assess Explain-in-Plain-English (EiPE) responses by generating executable code from students' explanations [17], and to generate personalised Parsons problems with customised contexts and concepts [18]. However, these approaches target general-purpose programming languages and do not address the unique demands of declarative query languages such as SQL.

Prior research on automatic SQL generation - whether for exercise creation [19, 20] or for content development with LLMs [21] - focuses on producing syntactically valid queries rather than probing relational reasoning. Similarly, automated SQL assessment tools [22, 23] compare student queries to

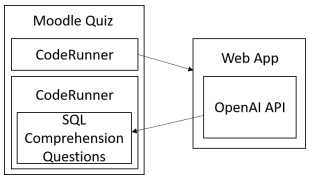

Figure 1: System architecture

model solutions or generate structural hints, but do not generate comprehension questions aimed at revealing misconceptions.

Together, these strands of research indicate that while foundations exist for SQL generation, autograding, and automatic question generation, no prior work generates SQL-specific comprehension questions tailored to an individual student's own query. This gap motivates our contribution: an extension of the AutoMCQ approach to SQL, capable of producing multiple-choice questions aligned with a student's submitted query and designed to surface SQL-specific misconceptions.

3. Approach

We previously developed a web application, AutoMCQ [11, 12], which uses GPT-4o mini via the OpenAI API to generate personalised multiple-choice code comprehension questions, based on a student's submitted code. The code for this tool is available on GitHub [1]. We integrated calls to this application within CodeRunner [24] quiz questions hosted on our Virtual Learning Environment (VLE), which is built on top of Moodle [2]. CodeRunner [3] is a Moodle plugin which allows users to submit code to be run against predefined test cases and can support multiple programming languages. The high level architecture of our approach can be seen in Figure 1. CodeRunner is not required to generate the code comprehension questions, however, it is used in our approach to allow easy integration within our VLE and our existing automated marking approach, which uses CodeRunner.

The students are presented with a traditional CodeRunner question (see [10] for more detail - similar questions can be developed for SQL [4]). When they then progress to the code comprehension questions their code plus other parameters are passed to our web application. This paper focusses on our extension of AutoMCQ to be applicable to SQL comprehension.

The prompt sent to the OpenAI API consists of the system prompt "You are an educational assistant specializing in computer science. Your task is to analyse students' SQL code for the beginner database class and generate thoughtful multiple-choice questions that can help them understand and improve their SQL skills. You should try and make good distractor options to really test students' understanding" [5]. In addition to the system prompt, the model receives parameters specifying: (i) the number of questions to generate, (ii) the CodeRunner question text, (iii) target SQL topics, (iv) the database schema (expressed in SQL), and (v) the student's submitted query. This then returns multiple-choice SQL comprehension questions and displays them within a CodeRunner question. The students can then answer these questions and have them automatically marked. Unfortunately, we can't rely on GenAI generating correct or sensible questions every time. Therefore, to handle this there is a note above the generated questions: "These questions were generated by AI. Therefore, questions generated may be incorrect. If you think they are incorrect please select 'This question doesn't seem right'. Also, select this option if the question doesn't relate to SQL." This option can be used to trigger a manual check by the class lecturer. Until GenAI models improve our SQL comprehension questions will only be used for formative assessment.

[1] https://github.com/RobbieBooth/AI-Question-Generator
[2] https://moodle.org/
[3] https://coderunner.org.nz/
[4] https://coderunner.org.nz/mod/page/view.php?id=551
[5] Multiple prompts were experimented with during development and this prompt was found to provide the most consistently usable questions.

Costing was considered to determine whether or not this is financially viable for large class sizes. Cost per query varies depending on the length of the submitted query and the number of questions generated, however, at the time of writing, questions tended to cost approximately $0.0002, making the approach financially viable even for large cohorts.

3.1. Examples

To evaluate the effectiveness of our SQL comprehension tool, we selected representative examples derived from the SQL tasks used by Miedema et al. [1], specifically those listed in Table 3 of their study. Although these tasks span several SQL concepts, the authors' qualitative analysis indicates that joins and aggregation with GROUP BY are the concepts most consistently associated with novice misconceptions in the think-aloud data, whereas simpler SELECT queries and ORDER BY tend to involve fewer conceptual difficulties. As the original study does not report quantitative frequencies, this distinction reflects qualitative prominence rather than a strict ranking. We therefore focused our evaluation on these two concepts, as they present the greatest opportunity to assess whether the tool can surface, explain, and support reasoning about non-trivial misunderstandings, which is central to its intended pedagogical use. The specific questions adapted from Miedema et al [1] that were used in our evaluation are listed in Table 1 and use the same schema, which is detailed in Table 2.

Table 1
SQL questions

#	Question	Covered concepts
1	List all pairs of customer IDs who live on a street with the same name but in a different city.	Self-join
2	List all customer IDs, dates and quantities of transactions containing products named Apples.	Natural join
3	Return a list of the number of stores per city.	Group By

Each question was implemented in CodeRunner using the SQLite database engine. For each question, the CodeRunner tests executed the submitted queries against tables populated with varying data values and compared the resulting outputs with the expected answers, ensuring that correctness was evaluated across multiple data instances rather than a single fixed dataset. This prevents students from gaming the system through hard-coded or printed outputs instead of correct query semantics.

For example, for **Q1**, one of the unit tests evaluated the query results when the customer table contained the data shown in Table 3. For the correct query (as shown in Query **1(a)** in Table 4), the following result will be returned:

The submitted SQL is then sent to the web application along with number of questions 2, the question, the topic "Self-join", the schema expressed as SQL queries, and the student's SQL query. For each question, the topic is set as the concept covered for the question in Table 1.

3.1.1. SQL Code Comprehension Questions

Examples of the generated SQL comprehension questions are shown in Table 4. For each question in Table 1, comprehension questions were generated from a correct reference query (denoted **(a)**) together with examples of incorrect queries (denoted **(b)** and **(c)**) taken directly from Miedema et al. [1]. For each query, we generated 20 questions in total, which were then reviewed by the authors to evaluate their quality. The authors focused on basic appropriateness, including clarity, relevance, and correctness. Overall, the review indicated that most questions satisfied these criteria. The remainder of this subsection discusses how the example comprehension questions probe students' understanding of SQL.

Table 2
The database schema used for the questions in Table 1. Underlined attributes collectively form the primary key for each table.

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

cID	cID
0	3
1	3

Table 3
Sample data for the Customer table

cID	cName	street	city
0	Martin	High Street	Glasgow
1	Alasdair	High Street	Glasgow
2	Andrew	George Street	Edinburgh
3	Robbie	High Street	Edinburgh
4	Clemens	Duke Street	Glasgow

Table 4: SQL statement and corresponding multiple-choice comprehension questions - correct answers marked in bold. The number in Query refers to the question number in Table 1.

Query	SQL Statement	Multiple-Choice Comprehension Question

1 (a)	`SELECT a.cID, b.cID`
	`FROM customer a`
	`JOIN customer b`
	`  ON a.street = b.street`
	`AND a.city <> b.city`
	`AND a.cID < b.cID;`
		Q: Why is `a.city <> b.city` included in the self-join condition?
		(a) To include only customers from the same city.
		(b) To ensure that the paired customers are from different cities.
		(c) To return customers regardless of their city.
		(d) To filter out customers who live on the same street.

1 (b)	`SELECT customer AS a, customer AS b`
	`WHERE a.street = b.street`
	`AND a.city <> b.city;`
		Q: Which SQL clause is incorrectly formatted in the provided code?
		(a) The WHERE clause should not use aliases.
		(b) The AND condition needs to be replaced with OR.
		(c) The SELECT clause should specify the columns to select, not the table name.
		(d) All of the above clauses are correctly formatted.

1 (c)	`SELECT cID`
	`FROM customers`
	`WHERE street = street`
	`AND city <> city;`
		Q: What is the correct syntax for referencing the same column in a self-join condition?
		(a) Use table aliases to distinguish between the two instances of the column.
		(b) Use a single table reference without aliases.
		(c) Use different column names for each instance.
		(d) Reference the column using only its name without specifying the table.

Query	SQL Statement	Multiple-Choice Comprehension Question

2 (a)	`SELECT cID, date, quantity`
	`FROM purchase`
	`NATURAL JOIN product`
	`WHERE pName = 'Apples';`
		Q: What is the purpose of using NATURAL JOIN in the student's SQL query?
		(a) To restrict the results to only matching records in both tables.
		(b) To create a subquery that rearranges the results based on product names.
		(c) To join all columns from both tables regardless of matching names.
		(d) To automatically join tables based on columns with the same name.

2 (b)	`SELECT transaction.cID, transaction.date,`
	`  transaction.quantity`
	`FROM transaction`
	`JOIN transaction.pID = product.pName`
	`WHERE pnamed = "Apples";`
		Q: In the student's SQL code, what is the error in the JOIN clause?
		(a) The JOIN should be a natural join instead of an inner join.
		(b) The JOIN should not include the pID at all.
		(c) The correct syntax should use ON instead of = in the JOIN clause.
		(d) The JOIN clause is correct and does not need any changes.

2 (c)	`SELECT s.cID, s.date, s.quantity`
	`FROM shoppinglist s, product p`
	`WHERE p.pName = "Apples";`
		Q: What is the main issue with the query when trying to retrieve transactions containing products named Apples?
		(a) The query does not include a join condition to link the 'shoppinglist' and 'product' tables.
		(b) The query uses the wrong delimiter for string values.
		(c) The query selects the wrong columns.
		(d) The query does not filter out products that are not in stock.

3 (a)	`SELECT city, COUNT(*) AS num_stores`
	`FROM store`
	`GROUP BY city;`
		Q: What will be the output of the COUNT(*) function in the context of the given SQL code?
		(a) The total number of stores for each city.
		(b) The average number of stores across all cities.
		(c) The number of unique cities in the store table.
		(d) The total inventory of all items in the store.

3 (b)	`SELECT city, COUNT(city)`
	`FROM store`
	`GROUP BY city;`
		Q: What is the purpose of using GROUP BY in SQL?
		(a) To filter rows based on a condition.
		(b) To aggregate data across specified columns.
		(c) To join multiple tables together.
		(d) To create a new table from the results.

3 (c)	`SELECT city, COUNT(city)`
	`FROM store`
	`GROUP BY city;`
		Q: In the query, what will happen if there are entries in the store table without a city specified?
		(a) Those entries will be included with a null city.
		(b) The query will fail due to invalid data.
		(c) Those entries will be excluded from the results.
		(d) The city will be counted as a blank entry.

Q1 - Self-join

Q1(a) shows an example of a generated SQL comprehension question based on a correct solution to **Q1**. Despite the query being correct, students may arrive at such solutions by relying on example-driven or trial-and-error approaches, modifying previously seen queries until they work rather than reasoning about SQL semantics, thereby reflecting the generalisation-based misconceptions described by Miedema et al. [1]. As a result, this

question focuses on the student's understanding of `a.city <> b.city`, specifically whether they recognise that `<>` denotes inequality. Such operators can be a source of confusion for novices, as students often transfer expectations from general-purpose programming languages - where `!=` is more common - to SQL, a pattern consistent with the language-based and prior-knowledge misconceptions identified by Miedema et al. [1].

Q1(b) presents an incorrect query that misuses the SELECT clause by specifying table aliases rather than columns. This question assesses whether students understand the role of the SELECT clause and can distinguish between table references and the attributes to be returned, reflecting common generalisation-based misconceptions.

Q1(c) relates to understanding of aliasing in self-joins. The query incorrectly compares columns without distinguishing between table instances, and the question assesses whether students recognise the need for aliases to disambiguate column references, aligning with previously identified language-based misconceptions.

Q2 - Natural join

Q2(a) focuses on the student's understanding of the semantics of NATURAL JOIN, specifically whether they recognise that it automatically joins tables based on columns with identical names. This question probes whether students understand how join conditions are inferred implicitly in SQL, rather than assuming that NATURAL JOIN behaves like a generic inner join. Such misunderstandings can arise when students generalise join behaviour from previously seen examples without reasoning about the role of column names.

Q2(b) focuses on the student's understanding of the correct syntax for explicit joins in SQL. The query incorrectly specifies a join condition using assignment syntax rather than an ON clause, and the question assesses whether students can distinguish between join structure and conditional expressions. Errors of this kind may occur when students transfer expectations from other programming contexts or rely on surface-level patterns rather than clause semantics.

Although the conceptual focus of this question should be on natural joins, the generated questions typically first target syntactic errors before addressing semantic or conceptual issues. When multiple questions are generated, these also include items that probe students' understanding of the definition of natural joins and their ability to rewrite the query using NATURAL JOIN.

Q2(c) focuses on the student's understanding of how tables must be linked when querying across multiple relations. The query references multiple tables but omits a join condition, resulting in an unintended Cartesian product. This question assesses whether students recognise the necessity of explicitly relating tables in SQL, a difficulty that can arise when students reason about queries procedurally rather than in terms of relational semantics.

Q3 - Group By

Q3(a) focuses on the student's understanding of aggregation in SQL, specifically the behaviour of the COUNT() function when used in conjunction with GROUP BY. This question assesses whether students recognise that COUNT() returns the number of rows within each group - in this case, the total number of stores per city, rather than a global count or an aggregate across all groups. Misunderstandings here often arise when students fail to reason about how aggregation operates over grouped data.

Q3(b) focuses on the student's understanding of the role of the GROUP BY clause in SQL. The question probes whether students recognise that GROUP BY is used to group rows based on specified attributes, enabling functions such as COUNT to be applied per group rather than across the entire table. Confusion around GROUP BY is common when students conflate aggregation with filtering or joining operations.

Q3(c) is based on the same query as **Q3(b)** and focuses on the student's understanding of how aggregate functions interact with NULL values. By using `COUNT(city)` rather than `COUNT(*)`, the query excludes rows where the `city` attribute is NULL. This question assesses whether students recognise that `COUNT(column)` only counts non-NULL values, a subtle but important aspect of SQL aggregation semantics that is often misunderstood by novices.

Together, these questions illustrate how the generated SQL comprehension questions assess students' understanding beyond surface-level syntactic correctness.

4. Conclusions and Future Work

We present an automated system for generating SQL comprehension questions within the CodeRunner ecosystem. We have demonstrated that our tool can generate coherent, query specific multiple-choice questions based on

student submitted SQL at low cost, making it feasible for use as a scalable formative assessment tool. While this represents a promising first step, several limitations and opportunities for further work remain.

Future work should include an empirical evaluation of the quality and educational effectiveness of the generated questions. Although this work demonstrates that the system can produce coherent SQL comprehension questions, a more in-depth evaluation is required to further assess question quality, and it remains to be studied whether students find these questions helpful for developing conceptual understanding. This may also involve further refinement of the prompt. Such an evaluation could also help determine the optimal number of comprehension questions to generate per query, as SQL queries are information dense and may present diminishing returns beyond a certain point.

Another limitation concerns the reliability of generative models. While our current mitigation strategy - allowing students to flag questions that appear incorrect or irrelevant - provides a pragmatic safeguard, it is not ideal. Improvements in model reliability, alongside additional validation or filtering mechanisms, represent important directions for future work.

While some limitations remain, we believe that our tool has the potential to fill an existing gap in the literature. We do not believe that this tool can replace the support that teaching staff can provide, however, it can be used to identify areas of weak understanding and provide a useful study tool in a scalable and cost-effective manner, reducing the manual effort required for instructors to author and deploy SQL comprehension questions.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools [6].

References

[1] D. Miedema, E. Aivaloglou, G. Fletcher, Identifying sql misconceptions of novices: Findings from a think-aloud study, in: Proceedings of the 17th ACM Conference on International Computing Education Research, ICER 2021, Association for Computing Machinery, New York, NY, USA, 2021, p. 355–367. URL: https://doi.org/10.1145/3446871.3469759. doi:10.1145/3446871.3469759.

[2] T. Lehtinen, A. Lukkarinen, L. Haaranen, Students struggle to explain their own program code, in: Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1, ITiCSE '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 206–212. URL: https://doi.org/10.1145/3430665.3456322. doi:10.1145/3430665.3456322.

[3] D. Miedema, G. Fletcher, E. Aivaloglou, 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, ICPC '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 122–132. URL: https://doi.org/10.1145/3524610.3529158. doi:10.1145/3524610.3529158.

[4] D. Miedema, G. Fletcher, E. Aivaloglou, Expert perspectives on student errors in sql, ACM Trans. Comput. Educ. 23 (2022). URL: https://doi.org/10.1145/3551392. doi:10.1145/3551392.

[5] C. Schulte, T. Clear, A. Taherkhani, T. Busjahn, J. H. Paterson, An introduction to program comprehension for computer science educators, in: Proceedings of the ITiCSE-WGR'10 - 2010 Working Group Reports on Innovation and Technology in Computer Science Education, Association for Computing Machinery, New York, NY, USA, 2010, pp. 65–86. URL: https://dl.acm.org/doi/pdf/10.1145/1971681.1971687. doi:10.1145/1971681.1971687, copyright 2010 ACM.

[6] J. Sheard, M. Dick, S. Markham, I. Macdonald, M. Walsh, Cheating and plagiarism: perceptions and practices of first year it students, SIGCSE Bull. 34 (2002) 183–187. URL: https://doi.org/10.1145/637610.544468. doi:10.1145/637610.544468.

[7] Simon, J. Sheard, M. Morgan, A. Petersen, A. Settle, J. Sinclair, G. Cross, C. Riedesel, Negotiating the maze of academic integrity in computing education, in: Proceedings of the 2016 ITiCSE Working Group Reports, ITiCSE '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 57–80. URL: https://doi.org/10.1145/3024906.3024910. doi:10.1145/3024906.3024910.

[8] P. Denny, A. Luxton-Reilly, E. Tempero, J. Hendrickx, Codewrite: supporting student-driven practice of java, in: Proceedings of the 42nd ACM Technical Symposium on Computer Science Education, SIGCSE '11, Association for Computing Machinery, New York, NY, USA, 2011, p. 471–476. URL: https://doi.org/10.1145/1953163.1953299. doi:10.1145/1953163.1953299.

[6] The only exception is the use of the OpenAI API to generate the SQL comprehension questions.

[9] A. Luxton-Reilly, E. Tempero, N. Arachchilage, A. Chang, P. Denny, A. Fowler, N. Giacaman, I. Kontorovich, D. Lottridge, S. Manoharan, S. Sindhwani, P. Singh, U. Speidel, S. Stephen, V. Terragni, J. Whalley, B. Wuensche, X. Ye, Automated assessment: Experiences from the trenches, in: Proceedings of the 25th Australasian Computing Education Conference, ACE '23, Association for Computing Machinery, New York, NY, USA, 2023, p. 1–10. URL: https://doi.org/10.1145/3576123.3576124. doi:10.1145/3576123.3576124.

[10] M. Goodfellow, A. Abel, K. Liaskos, J. Levine, Automated marking in undergraduate programming classes, in: Proceedings of the 8th Conference on Computing Education Practice, CEP '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 13–16. URL: https://doi.org/10.1145/3633053.3633060. doi:10.1145/3633053.3633060.

[11] M. Goodfellow, R. Booth, A. Fagan, A. Lambert, Automcq - automatically generate code comprehension questions using genai, in: Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 2, ITiCSE 2025, Association for Computing Machinery, New York, NY, USA, 2025, p. 737–738. URL: https://doi.org/10.1145/3724389.3731266. doi:10.1145/3724389.3731266.

[12] M. Goodfellow, R. Booth, A. Fagan, A. Lambert, Testing code comprehension using genai, in: Proceedings of the 2025 Conference on UK and Ireland Computing Education Research, UKICER '25, Association for Computing Machinery, New York, NY, USA, 2025. URL: https://doi.org/10.1145/3754508.3754532. doi:10.1145/3754508.3754532.

[13] K. Presler-Marshall, S. Heckman, K. T. Stolee, Sqlrepair: identifying and repairing mistakes in student-authored sql queries, in: Proceedings of the 43rd International Conference on Software Engineering: Joint Track on Software Engineering Education and Training, ICSE-JSEET '21, IEEE Press, 2021, p. 199–210. URL: https://doi.org/10.1109/ICSE-SEET52601.2021.00030. doi:10.1109/ICSE-SEET52601.2021.00030.

[14] T. Taipalus, M. Siponen, T. Vartiainen, Errors and complications in sql query formulation, ACM Trans. Comput. Educ. 18 (2018). URL: https://doi.org/10.1145/3231712. doi:10.1145/3231712.

[15] A. Migler, A. Dekhtyar, Mapping the sql learning process in introductory database courses, in: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, SIGCSE '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 619–625. URL: https://doi.org/10.1145/3328778.3366869. doi:10.1145/3328778.3366869.

[16] S. Shin, Effect of semantic distance on learning structured query language: An empirical study, Frontiers in Psychology 13 (2022) 996363. URL: https://doi.org/10.3389/fpsyg.2022.996363. doi:10.3389/fpsyg.2022.996363.

[17] P. Denny, D. H. Smith, M. Fowler, J. Prather, B. A. Becker, J. Leinonen, Explaining code with a purpose: An integrated approach for developing code comprehension and prompting skills, in: Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1, ITiCSE 2024, Association for Computing Machinery, New York, NY, USA, 2024, p. 283–289. URL: https://doi.org/10.1145/3649217.3653587. doi:10.1145/3649217.3653587.

[18] A. del Carpio Gutierrez, P. Denny, A. Luxton-Reilly, Automating personalized parsons problems with customized contexts and concepts, in: Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1, ITiCSE 2024, Association for Computing Machinery, New York, NY, USA, 2024, p. 688–694. URL: https://doi.org/10.1145/3649217.3653568. doi:10.1145/3649217.3653568.

[19] D. R. Quan Do, Rajeev K. Agrawal, V. N. Gudivada, Automatic generation of sql queries, in: 2014 ASEE Annual Conference & Exposition, 10.18260/1-2–20112, ASEE Conferences, Indianapolis, Indiana, 2014. Https://peer.asee.org/20112.

[20] S. Abdul Khalek, S. Khurshid, Automated sql query generation for systematic testing of database engines, in: Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering, ASE '10, Association for Computing Machinery, New York, NY, USA, 2010, p. 329–332. URL: https://doi.org/10.1145/1858996.1859063. doi:10.1145/1858996.1859063.

[21] W. Aerts, G. Fletcher, D. Miedema, A feasibility study on automated sql exercise generation with chatgpt-3.5, in: Proceedings of the 3rd International Workshop on Data Systems Education: Bridging Education Practice with Education Research, DataEd '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 13–19. URL: https://doi.org/10.1145/3663649.3664368. doi:10.1145/3663649.3664368.

[22] S. Nayak, R. Agarwal, S. K. Khatri, M. Mohammadian, Student outcome assessment on structured query language using rubrics and automated feedback generation, International Journal of Advanced Computer Science and Applications (2024). URL: https://api.semanticscholar.org/CorpusID:268859376.

[23] C. Kleiner, F. Heine, Enhancing feedback generation for autograded sql statements to improve student learning, in: Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1, ITiCSE 2024, Association for Computing Machinery, New York, NY, USA, 2024, p. 248–254. URL: https://doi.org/10.1145/3649217.3653579. doi:10.1145/3649217.3653579.

[24] R. Lobb, J. Harlow, Coderunner: A tool for assessing computer programming skills, ACM Inroads 7 (2016)

47–51. URL: https://doi.org/10.1145/2810041. doi:10.1145/2810041.