A Proposal for Revising SQL Error Taxonomies Based on Automated Detection

Davide Ponzini*, Giovanna Guerrini and Barbara Catania

University of Genoa, Italy

Abstract

Several research efforts have addressed the problem of characterizing student mistakes in database query writing, among which the SQL error taxonomy proposed by Taipalus et al. [1] has become a widely adopted framework. In this paper, we report ongoing work stemming from our experience in operationalizing this taxonomy within an automated SQL error-detection pipeline, where we encountered practical limitations and ambiguities arising when taxonomy categories are interpreted algorithmically rather than by human annotators. Building on extensive manual annotation and iterative refinement conducted during the development of an automated SQL error correction tool, we outline a practice-driven revision of the taxonomy aimed at supporting fine-grained automated error classification. The proposed revision focuses on refining error definitions, regrouping certain categories, clarifying labels, and addressing gaps and redundancies identified in the original taxonomy. We conclude by discussing how the revised taxonomy can support educational tools and by outlining future directions for validation and empirical assessment.

1. Introduction

Data systems education represents a challenging component of many computer science degree programmes. While research on query language education has recently gained increased attention, it remains less developed than research on programming language education. Declarative query languages such as SQL may appear intuitive, yet students frequently struggle to master them. A well-documented source of difficulty lies in the contrast between SQL's declarative, set-based foundations and the imperative, instance-based paradigms typically encountered in CS1 courses (see, e.g., [2, 3, 4]). Although database courses usually include hands-on interaction with data systems, existing debugging facilities and error messages provide limited pedagogical support, as they often fail to clearly communicate the nature of students' mistakes [4, 5, 6].

Research on SQL education has produced valuable results, including taxonomies for classifying student errors [1] and analyses of the misconceptions underlying common mistakes [2]. These approaches aim to support instructors and learners through more informative feedback, but they rely heavily on manual expert annotation and, in some cases, think-aloud studies. In parallel, other work has focused on automated support tools, ranging from query correction [7] to hint generation and scaffolding [8].

Our work builds on these contributions through the development of the LenSQL tool [9], which employs error taxonomies as the backbone for automated feedback, learning analytics, and personalised exercise generation. During its development, we adopted the taxonomy proposed in [1], which categorises SQL errors into syntax errors, semantic errors (queries that are semantically incorrect regardless of the request), logical errors (queries that are semantically correct but do not match the request), and complications (queries that return the correct result but are overly complex). While comprehensive, this taxonomy was originally designed for expert human annotators. When operationalised within an automated error-detection pipeline, several practical limitations emerged, including ambiguities in category interpretation and difficulties in translating definitions into algorithmic criteria. To the

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

*Corresponding author.
davide.ponzini@edu.unige.it (D. Ponzini); giovanna.guerrini@unige.it (G. Guerrini); barbara.catania@unige.it (B. Catania)
0009-0006-4282-9652 (D. Ponzini); 0000-0001-9125-9867 (G. Guerrini); 0000-0002-6443-169X (B. Catania)

best of our knowledge, no alternative taxonomy has been explicitly designed to support fine-grained, automated SQL error classification.

This paper presents ongoing research aimed at addressing this gap by revising the taxonomy in [1] to make it suitable for automated use while preserving its pedagogical intent. The revision is grounded in extensive manual annotation and iterative refinement carried out during the development of our error detector. It is guided by three main principles: (i) more precise definitions of syntax, semantic, and logical errors, as well as complications; (ii) reorganisation of error types to better align with these definitions; and (iii) clearer labels to reduce ambiguity. The revision also removes redundant or insufficiently distinct categories and introduces new ones to cover previously unaddressed scenarios.

The revised taxonomy aims to bridge educational research and intelligent support systems. By grounding error categories in machine-detectable properties while retaining their educational meaning, it supports reliable automatic classification, targeted feedback generation for students, and aggregated analyses that can inform instructors and, in future work, support reasoning about underlying misconceptions. At the same time, the taxonomy remains interpretable for researchers and educators conducting manual analyses of student performance. Thus, we strengthen the pedagogical value of error classification by making categories actionable for targeted feedback, personalized tutoring, and data-driven exercise generation.

The contributions of this paper are as follows: (i) a systematic, practice-driven revision of an established SQL error taxonomy to support fine-grained automated classification; (ii) refinement and reorganisation of error categories to remove ambiguities and redundancies while introducing machine-detectable distinctions; and (iii) the definition of a framework that connects educational research with automated assessment tools, enabling actionable feedback and data-driven support.

The remainder of the paper reviews related work, outlines the requirements and principles guiding the revision, presents preliminary revisions to the taxonomy, and discusses ongoing and future research directions.

2. Related Work

In this section, we briefly discuss the relevant related work in SQL error classification, identification of frequent students misconceptions, and automatic correction of SQL queries.

Analyzing and Classifying Students Errors in Formulating SQL Queries From syntactic errors to more advanced semantic and logical misinterpretations, errors in SQL queries can take many forms. The idea of analyzing and classifying (frequent) student errors is not new: Brass and Goldberg [10] identify categories of queries that are syntactically correct but certainly not intended (e.g, because they are contradictory), Ahadi et al. [11] carried on an extensive study on the mistakes of novices database query writers, in the same line as the extensive amount of studies devoted to the mistakes of novice programmers. Other relevant analyses of common errors have been carried on by Taipalus et al. [1], Poulsen et al. [12], Presler et al. [13] (on the provision of repair strategies). The problem is also investigated, with a slightly different focus, in [14] with the aim of identifying SQL problems in data-intensive code. Taipalus et al. in [1] propose the quite comprehensive taxonomy that will be the basis of our study. As a starting point of our study, we have collected and classified according to this taxonomy the errors made by the students in our course in different settings [15].

Identifying Misconceptions Taipalus et al. [3] and Miedema et al. in [2, 16] focus on understanding the thinking aspects of learning SQL with the aim of getting the causes why students make mistakes so often. Miedema et al. [2] conducted a think-aloud research to highlight typical beginner mistakes and they collected and analyzed the hypotheses of SQL experts about the causes of student errors [17]. Misconceptions also arise from knowledge transfer from other domains, making learning SQL challenging [2]. Many of the identified challenges are due to the declarative nature of SQL [2, 3, 4] and the manipulation of data collections rather than individual instances, as in programming languages [18].

The procedural SQL-like language PRQL [19] has been proposed starting from the observation that procedural languages could be more intuitive to learn for novices, compared to declarative languages.

Automatic Feedback and Repair to SQL Queries Several approaches and prototypes have been designed to provide automatic feedback and correction for SQL teaching and learning. The tools aim to enhance learning outcomes by identifying errors, offering constructive feedback, and supporting self-guided improvement [20, 21]. Many approaches target automatic correction and grading of SQL queries. The support ranges from the integration of CodeRunner in a Learning Management System like Moodle for providing immediate feedback by executing the query formulated by the student [22], to automatic grading systems [7, 23]. The full potential of such tools in enhancing learning can only be leveraged, if they extend beyond grading efficiency by also providing tutoring capabilities to the students. Hint generation is, for instance, the focus of [8, 24]. Recently, approaches have been developed for automatic SQL correction based on Large Language Models [25], for instance fine-tuned GPT models are used to detect and provide feedback on semantic errors in SQL queries in [26].

3. Methodology

The methodology adopted in this work is grounded in the practical task of implementing an automated SQL error detector based on a widespread taxonomy. Rather than proposing theoretical refinements in isolation, the revision process emerged from a systematic, error-by-error operational analysis conducted during the development of the detector. This section describes the analytical procedure followed, the criteria used to identify limitations of the original taxonomy, and the principles guiding the proposed revisions.

3.1. Operationalization of the Taxonomy

As a first step, each error type in the reference taxonomy was translated into a set of conditions that an automated system could verify on a student query. This process required making explicit the assumptions underlying each error label, such as whether the query must fail to execute, whether comparison with a reference solution is required, or whether schema metadata inspection is necessary.

During this phase, several ambiguities between errors and student misconceptions became apparent. For instance, the syntax error "*restriction in SELECT clause (e.g., SELECT fee > 10)*" refers to a misconception about SELECT and WHERE clauses, rather than invalid SQL syntax (`SELECT fee > 10` returns a boolean value for each row). These cases motivated a shift from intent-based definitions (i.e., maybe the student is confused about the role of the clauses) to criteria grounded in observable query behavior (i.e., the query cannot be executed because of a syntax error).

3.2. Case-Driven Error Analysis

The detector was incrementally implemented by testing real and synthetic student queries against each error definition. For each query-error pairing, we assessed whether the taxonomy allowed for: (i) a clear and unambiguous classification, (ii) a feasible algorithmic detection strategy, and (iii) a meaningful interpretation in view of future mappings to student misconceptions.

This analysis led to the identification of three recurrent situations: (i) queries that could reasonably be assigned to multiple error categories depending on interpretation, (ii) queries for which no existing error category provided an adequate description, (iii) error categories whose definition hindered a principled alignment with misconceptions rather than query properties.

These situations were systematically documented and used as evidence for revision decisions.

3.3. Revision Principles

Based on the identified issues, revisions were guided by a set of methodological principles. First, precise naming criteria were adopted to prevent ambiguous error definitions. Second, error categories were redefined to rely exclusively on properties that can be derived from the query itself, optionally in relation to schema metadata or correct reference queries[1]. Third, overlapping or weakly differentiated categories were merged or re-grouped to reduce classification ambiguity.

Importantly, no revision was introduced solely to simplify implementation. Instead, changes were adopted only when the original formulation systematically hindered consistent detection or obscured the pedagogical interpretation of the error.

3.4. Implications for Automated Detection and Future Mapping

Although the present work focuses on taxonomy revision rather than misconception modeling, the methodology explicitly considers future mappings between errors and misconceptions. Categories that encode multiple conceptual failures or that depend heavily on inferred student intent were flagged as problematic for such mappings. By enforcing clearer, behavior-based definitions, the revised taxonomy is designed to act as a stable intermediate layer between raw student queries and higher-level cognitive interpretations, ensuring that it remains pedagogically meaningful while being suitable for automated error detection and scalable educational tools.

4. Taxonomy Revisions

A comprehensive and up-to-date version of the taxonomy is available online[2], where illustrative examples are provided for each error. Representative queries highlighting the relevant revisions to the taxonomy are reported in Table 1. The underlying schema, described in Table 2, is adapted from Miedema et al. [2].

4.1. Naming Criteria

We propose using the following naming criteria to reduce ambiguity in the error definitions:

Table Reference We suggest renaming "*join*" into "*table reference*" whenever we mean which tables are included in the FROM clause, since "*join*" fails to capture errors in the first element of a FROM clause. For example, if a student wrote `SELECT * FROM customer JOIN store ON ...` instead of `SELECT * FROM inventory JOIN store ON ...`, technically it is not a *join* on an incorrect table error, since the joined table is correct. However, this error can still be described as an incorrect *table reference*.

Join Condition We suggest renaming "*join*" into "*join condition*" whenever we mean a condition applied between columns of two different tables. This condition can be either in the `JOIN ... ON ...` clause or at the top level of the conjunctive normal form of the WHERE clause (i.e. under implicit join syntax). This clarifies whether "*join*" refers to the presence of a table in the FROM clause or to a condition between columns of these tables.

4.2. Syntax Errors

Ambiguous Columns We noticed that error #1 "*omitting correlation names*", as discussed in [27] and [28], refers to referring to a column present in at least two tables referenced in the query without specifying which table the column belongs to. Similarly, error #2 "*ambiguous column*", as discussed

[1]We assume that (at least) one correct reference query for each data demand is available, which is reasonable in our intended usage scenarios.

[2]https://github.com/DavidePonzini/sql_error_taxonomy

ID	SQL statement	Request	Original taxonomy	Revised taxonomy
1	SELECT pID, P unit_price > 10 FROM inventory;	Select all items IDs and whether they have a unit price greater than 10.	28. Restriction in SELECT clause	(No errors)
		Select the IDs of items have a unit price greater than 10.	28. Restriction in SELECT clause	70. Extraneous column in SELECT 66. Missing expression
2	SELECT street FROM customer GROUP BY street;	Select all street names without duplicates	31. Confusing the logic of keywords	97. GROUP BY can be replaced with DISTINCT
		Select all street names in alphabetical order	31. Confusing the logic of keywords	*Extraneous GROUP BY* *Missing ORDER BY*
3	SELECT street, COUNT(*) FROM customer HAVING street;	Select how many customers live in each street	31. Confusing the logic of keywords	13. Data type mismatch 17. Strange HAVING
4	SELECT * FROM inventory WHERE unit_price >= (SELECT unit_price FROM product NATURAL JOIN inventory WHERE pName = 'Banana');	(Not relevant)	(No errors)	*Missing quantifier*
5	SELECT city, street FROM store INTERSECT SELECT city FROM customer;	(Not relevant)	(No errors)	*Different tuples in set operation*
6	SELECT * FROM store AS store;	(Not relevant)	(No errors)	*Correlation name identical to table name*

Table 1
Examples of SQL statements and corresponding categorization in the taxonomy. Categories in italic have been introduced in this revision.

Table name	Attributes
customer	cID, cName, street, city
store	sID, sName, street, city
product	pID, pName, suffix
inventory	sID, pID, date, quantity, unit_price

Table 2
Database schema used for the queries in Table 1. Underlined attributes collectively form the primary key for each table.

in [11], refers to the SQL error code 42702, namely "*ambiguous column*", which is caused by the same problem. We believe having two distinct errors for the same issue can only lead to confusion, and we suggest merging these two errors into a single error called "*ambiguous column*", which can be defined as "*referring to a column present in at least two tables referenced in the FROM clause, without specifying which table the column belongs to*".

Restriction in SELECT Clause Error #28 "*restriction in SELECT clause*" corresponds to a query which is actually syntactically correct. A SQL query can use operators and comparisons in the SELECT clause: the returned value is the result of the expression. For instance, using the example provided in [1], `SELECT fee > 10` will return `true` or `false`, depending on the value of each row. Of course, this may not match the data demand, but in this case the error will be logical, not syntactical, as shown in query 1.

Projection in WHERE Clause Error #29 "*projection in WHERE clause*" refers to a student misconception, rather than to an actual syntax error. We suggest using error #32 "*confusing the syntax of keywords*" for this error, since the problem can be described as incorrect syntax for the WHERE clause, since it should be provided with an expression that can be evaluated to a Boolean.

Confusing the Logic of Keywords Detecting error #31 "*confusing the logic of keywords*" can be very complex, even for a human, and does not always result in a syntax error. For instance, query 2 executes correctly, and as such presents no syntax errors. As shown in the example, categorization depends heavily on the data demand. On the other hand, query 3 cannot be executed, since the condition in the HAVING clause cannot be evaluated to a Boolean value. We can argue that the user could have meant to group by street instead of applying a HAVING condition on a string, but without further context (which is not needed for detecting syntax errors), we have no way of knowing for sure. Our suggestion is simply to apply the other categories, focusing on the actual query, instead of a supposed user intention.

Subquery Errors We suggest creating a new category, called "*Subquery errors*", that contains error #26 "*too many columns in subquery*" and a new error: "*missing quantifier*", which refers to not using a quantifier (ANY/ALL) with a subquery that could return more than one row. Consider query 6: if the join between `product` and `inventory` contains at most one `'Banana'`, the query can execute successfully, otherwise the DBMS throws an error. Even though the query is not necessarily syntactically incorrect, tagging a query with this error if a comparison is performed without a quantifier and the subquery can potentially return multiple rows (i.e., the subquery is missing a condition on primary/unique keys) would prevent potential future execution errors (i.e., upon changes in the database instance).

Curly, Square, or Unmatched Brackets We suggest splitting error #34 "*curly, square or unmatched brackets*" into two more detailed errors, namely "*curly or square brackets*" and "*unmatched brackets*". These errors reflect different misconceptions in students: the former means that students are not aware that they can only use parentheses in SQL, while the latter could be, for example, attributed to distraction.

IS where not Applicable We suggest moving error #35 "*IS where not applicable*" from *SYN-6 Common syntax error* to *SYN-3 Data type mismatch*, since using the IS operator expects either NULL or a Boolean value. Any other value, e.g., a string, triggers a data type mismatch, and as a consequence, we believe this category to be more appropriate.

Different Tuples in Set Operation We suggest adding a new error, namely *SYN-6 (Common Syntax Error): different tuples in set operation.* This syntax error is caused by having two queries that return tuples with different cardinalities combined with a set operation (UNION/ INTERSECT/EXCEPT). For example, query 9 is invalid, since the first SELECT returns two columns, while the second only one.

4.3. Semantic Errors

AND instead of OR (Empty Result Table) An invalid AND condition which always results in an empty result table is already accounted for in error #40 "*implied, tautological or inconsistent expression*". Moreover, a valid condition which uses AND instead of OR and does not result in an empty result table (i.e., non mutually-exclusive conditions) is not accounted for in the taxonomy (the only similar error is "*using OR instead of AND*", but this is the opposite scenario). In light of this, we suggest renaming this error to a generic "*AND instead of OR*", independently from the cardinality of the result set, and considering it a logical error.

Wildcards without LIKE We suggest considering error #43 as logical, since we cannot determine *a priori* if a user actually meant to equate a string to a wildcard character.

Incorrect Wildcard Similarly, we suggest considering error #44 as logical, since using a wildcard character in a string equation is not always wrong. Additionally, we suggest splitting this error into two more detailed errors, namely "*wrong wildcard*" and "*invalid wildcard*". The former error refers to using '_' instead of '%', or vice-versa, and highlights difficulty in understanding which wildcard to use. The latter refers to using characters which are interpreted as wildcards in common pattern matching languages but are treated as regular characters by the LIKE operator (e.g. using '*' instead of '%', or '?' instead of '_').

Omitting a Join We suggest considering error #48 as logical, since Cartesian products between tables are supported by the SQL standard and may be desirable in some scenarios. However, by knowing the data demand, we can easily determine whether a Cartesian product is expected by the data demand. Additionally, we suggest renaming this error to "*missing join condition*", to better align with our naming convention.

4.4. Logic Errors

Putting NOT in Front of Incorrect IN/EXISTS Error #56 is a subset of errors #53 "*extraneous NOT operator*" and #54 "*missing NOT operator*". This error is more specifically related to queries that have two nested subqueries that use IN/EXISTS operators. Since, in our experience, this is a pretty niche scenario, we believe no further distinction is needed. Additionally, the error definition is not entirely clear: are at least two IN/EXISTS, one of which should be negated, necessary for this error? What if the query contains only a single IN/EXISTS operator or if both operators are supposed to be negated but only one is? To solve these problems, we suggest counting only if the NOT operator should (not) have been present in the logical expression.

Join when Join Needs to be Omitted We suggest renaming error #59 to *"Extraneous table reference"*, to better align it with the other error names.

Missing Table Reference We suggest renaming error #62 "*missing join*" to "*Missing table reference*", to better distinguish it from error #48.

Clause Errors We suggest adding a new error category for when a SQL clause is used when not needed, or vice-versa. This category contains the following errors:

- **Missing clause:** a SQL clause, which is required by the data demand, is not present at all. Not using a clause highlights a different misconception in students than using that clause incorrectly, and as such should be treated separately.
- **Extraneous clause:** a SQL clause not required by the data demand is present in the query. This scenario is only accounted for the ORDER BY clause (error #76), but we suggest considering all SQL clauses.
- **Incorrect limit:** a LIMIT clause is required by the data demand, and the value provided in the query is incorrect.
- **Incorrect offset:** an OFFSET clause is required by the data demand, and the value provided in the query is incorrect.

4.5. Complications

Correlation Names are Always Identical We suggest splitting error #86 into two more specific errors, to better align with the two possible interpretations of this error. On the one hand, as described by [1], this error refers to having two table references with always the same data, since they refer the same table and their keys are equated. We suggest calling this error "*tables have the same data*", to enhance clarity. On the other hand, as the error name suggests, it is also possible to alias a table with

the table name itself (see query 6). Such an alias has no effect and makes the query harder to read. We suggest calling this error "*correlation name identical to table name*".

Condition on Left Table in LEFT OUTER JOIN As the name suggests, error #104 does not consider conditions on the right table for `LEFT OUTER JOIN`s, as well as any condition in `RIGHT/FULL OUTER JOIN`s. Brass et al. state that "*conditions on the right table do make sense in the left outer join condition*" [10]. However, it can be argued that both cases strongly depend on the actual data demand. Additionally, other join types should be considered as well. For example, the following queries produce the same output, even though they use, respectively a `LEFT` and a `RIGHT OUTER JOIN`:

- `SELECT c.*, s.* FROM store s LEFT OUTER JOIN customer c ON s.city = c.city AND s.sname = 'Coop';`
- `SELECT c.*, s.* FROM customer c RIGHT OUTER JOIN store s ON s.city = c.city AND s.sname = 'Coop';`

First, we suggest expanding this error to consider all conditions in the `JOIN` ON clause that involve either the left or the right table, as well as considering all outer join operations (`LEFT`, `RIGHT`, `FULL`). Additionally, we also suggest considering this error logical, since it can affect the resulting data set.

Unused CTE We suggest adding a new complication for defining a CTE *(Common Table Expression)* in a query and then not referencing it anywhere else.

5. Conclusion & Future Work

In this paper, we presented a practice-driven proposal for revising the error taxonomy in [1] based on our experience developing an automated SQL error detection system. The proposed revision originated by extensive manual annotation and iterative refinement, that identified ambiguities, redundancies, and missing categories that hinder the direct computational application of the original taxonomy.

The reported work is ongoing, as further refinement is needed within the Logical Errors category, which was only partially addressed in this study. In particular, operator- and expression-related errors still exhibit conceptual overlap and boundary ambiguities. Errors involving redundant, missing, or inconsistent logical expressions are undergoing systematic revision grounded in formal properties of expressions and empirical observations from student queries.

Moreover, the revised taxonomy should be regarded as a grounded proposal rather than a consolidated or exhaustive classification. While the changes presented here are informed by concrete implementation experience, further work is needed to examine how the taxonomy is applied by human annotators in practice and how it performs in fully automated settings. Studying annotation behavior, sources of disagreement, and the interaction between manual and automatic annotation can guide iterative refinements of category definitions and guidelines. In this perspective, the creation of a publicly available benchmark of manually annotated SQL queries would be a valuable contribution, supporting reproducibility and comparative evaluation of future error detection approaches, and strengthening the taxonomy's reliability.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT in order to: Paraphrase and reword, Improve writing style, Abstract drafting. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] T. Taipalus, M. Siponen, T. Vartiainen, Errors and Complications in SQL Query Formulation, ACM Trans. on Computing Education 18 (2018) 1–29.

[2] D. Miedema, E. Aivaloglou, G. Fletcher, Identifying SQL Misconceptions of Novices: Findings from a Think-aloud Study, ACM Inroads 13 (2022) 52–65.

[3] T. Taipalus, Explaining Causes behind SQL Query Formulation Errors, in: 2020 IEEE Frontiers in Education Conference (FIE), IEEE, 2020, pp. 1–9.

[4] J. Yang, A. Gilad, Y. Hu, H. Meng, Z. Miao, S. Roy, K. Stephens-Martinez, What Teaching Databases Taught Us about Researching Databases: Extended Talk Abstract, in: Proc. of the 3rd Int'l Workshop on Data Systems Education, 2024, pp. 1–6.

[5] T. Taipalus, SQL: A Trojan Horse Hiding a Decathlon of Complexities, in: Proc. of the 2nd Int'l Workshop on Data Systems Education, 2023, pp. 9–13.

[6] T. Taipalus, H. Grahn, Framework for SQL Error Message Design: A Data-Driven Approach, ACM Trans. on Software Engineering and Methodology (2023).

[7] B. Chandra, B. Chawda, B. Kar, K. M. Reddy, S. Shah, S. Sudarshan, Data Generation for Testing and Grading SQL Queries, The VLDB Journal 24 (2015) 731–755.

[8] Y. Hu, A. Gilad, K. Stephens-Martinez, S. Roy, J. Yang, Qr-Hint: Actionable Hints Towards Correcting Wrong SQL Queries, Proc. of the ACM Conf. on Management of Data 2 (2024) 1–27.

[9] D. Ponzini, B. Catania, G. Guerrini, Enhancing SQL Learning Through Generative AI and Student Error Analysis, in: New Trends in Database and Information Systems, Springer, 2025, pp. 118–128.

[10] S. Brass, C. Goldberg, Semantic Errors in SQL Queries: A Quite Complete List, Journal of Systems and Software 79 (2006) 630–644.

[11] A. Ahadi, J. Prior, V. Behbood, R. Lister, Students' Semantic Mistakes in Writing Seven Different Types of SQL Queries, in: Proc. of the 2016 ACM Conf. on Innovation and Technology in Computer Science Education, 2016, pp. 272–277.

[12] S. Poulsen, L. Butler, A. Alawini, G. L. Herman, Insights from Student Solutions to SQL Homework Problems, in: Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education, 2020, pp. 404–410.

[13] K. Presler-Marshall, S. Heckman, K. Stolee, SQLRepair: Identifying and Repairing Mistakes in Student-authored SQL Queries, in: Proc. of the 43rd Int'l Conf. on Software Engineering: Software Engineering Education and Training, 2021, pp. 199–210.

[14] B. A. Muse, M. M. Rahman, C. Nagy, A. Cleve, F. Khomh, G. Antoniol, On the Prevalence, Impact, and Evolution of SQL Code Smells in Data-Intensive Systems, 2022. arXiv:2201.02215.

[15] D. Ponzini, A. Livani, G. Guerrini, B. Catania, M. Coccoli, Analyzing Common Student Errors in SQL Query Formulation to Enhance Learning Support, in: Proc. 33rd Symposium on Advanced Database Systems, CEUR Workshop Proceedings, 2025.

[16] D. E. Miedema, On Learning SQL: Disentangling concepts in Data Systems Education, Phd thesis, Eindhoven University of Technology, 2024.

[17] D. Miedema, G. Fletcher, E. Aivaloglou, Expert Perspectives on Student Errors in SQL, ACM Trans. Comput. Educ. 23 (2022).

[18] A. Migler, A. Dekhtyar, Mapping the SQL Learning Process in Introductory Database Courses, in: Proc. of the 51st ACM Techn. Symp. on Computer Science Education, 2020, pp. 619–625.

[19] PRQL, Pipelined Relational Query Language, https://prql-lang.org, 2024.

[20] C. Kenny, C. Pahl, Automated Tutoring for a Database Skills Training Environment, in: Proc. of the 36th Techn. Symp. on Computer Science Education, 2005, pp. 58–62.

[21] S. Brass, SQLLint: Prototype of a Semantic Checker for SQL Queries, https://github.com/stefanbrass/sqllint, GitHub Repository, 2005.

[22] A. Wójtowicz, M. Prill, Relational Database Courses with CodeRunner in Moodle: Extending SQL Programming Assignments to Client-Server Database Engines, in: Proc. of the 56th ACM Techn. Symp. on Computer Science Education, 2025, pp. 1239–1245.

[23] K. Manikani, R. Chapaneri, D. Shetty, D. Shah, SQL Autograder: Web-based LLM-powered

Autograder for Assessment of SQL Queries, International Journal of Artificial Intelligence in Education (2025) 1–31.
[24] C. Kleiner, F. Heine, Enhancing Feedback Generation for Autograded SQL Statements to Improve Student Learning, in: Proc. of the Int'l Conf. on Innovation and Technology in Computer Science Education, ITiCSE 2024, 2024, p. 248–254.
[25] Z. Chen, S. Chen, M. White, R. Mooney, A. Payani, J. Srinivasa, Y. Su, H. Sun, Text-to-SQL Error Correction with Language Models of Code (2023). arXiv:2305.13073.
[26] A. AlRabah, S. Yang, A. Alawini, Optimizing Database Query Learning: A Generative AI Approach for Semantic Error Feedback, in: ASEE Annual Conference and Exposition, Conference Proceedings, American Society for Engineering Education, 2024.
[27] C. Welty, Correcting User Errors in SQL, International Journal of Man-Machine Studies 22 (1985) 463–477.
[28] J. B. Smelcer, User Errors in Database Query Composition, International Journal of Human-Computer Studies 42 (1995) 353–381.