Enhancing SQL Learning through Generative AI and Student Error Analysis

Davide Ponzini[0009−0006−4282−9652], Barbara Catania[0000−0002−6443−169X0], and Giovanna Guerrini[0000−0001−9125−9867]

Università di Genova, Genova, Italy
davide.ponzini@edu.unige.it, {barbara.catania,giovanna.guerrini}@unige.it

Abstract. Despite the widespread use of SQL in both academic and professional contexts, students often struggle with writing correct queries due to a range of syntactic and semantic misconceptions. In this work, we propose a framework supporting SQL learning centered around errors as central to the learning process. The framework integrates generative AI and automated error categorization to foster metacognitive engagement and support personalization.

1 Introduction

Context. Learning SQL presents difficulties for students, both in terms of syntax and semantics, even in degrees in which database related courses play an essential role. Prior research has shown that student misconceptions in SQL can be persistent and difficult to correct without targeted interventions. Identifying and classifying students' errors in SQL queries [19] can be a first step to enable a data-driven approach [5], provide specific feedback, and develop resources that target specific student misconceptions. SQL learning can be enhanced by approaches and prototypes that identify errors, offering constructive feedback and supporting self-guided improvement [8]. The integration of generative AI in such frameworks can be extremely beneficial, as it offers unprecedented scalability, and, among many opportunities, can provide students with clearer feedback.
Contribution. Grounded on the above mentioned body of data system education work, and along the same lines of approaches for integrating generative AI in programming environments for educational use, such as approaches to fine-tune GPT models to detect and provide feedback on SQL errors [3, 11], we designed and are currently developing an SQL tutoring framework. The framework integrates a data-driven approach with interactive learning mechanisms, relying on generative AI, to provide structured feedback and encourage deeper understanding. More specifically, it: (i) logs and analyzes all student queries to identify common mistakes; (ii) provides an interactive AI-powered tutor that guides students through the error resolution process without offering direct solutions; (iii) generates personalized exercises that target specific weaknesses, leveraging students' interests to enhance engagement. These interventions aim to develop students' ability to reason about mistakes, while enabling instructors to better understand learning trajectories and provide tailored support.

The main novelty of our approach lies in combining previous work in more traditional areas of error analysis [19] and automatic feedback [7] with a principled integration of generative AI. Our goal is to mitigate the risks associated with the use of AI from two perspectives: minimizing the impact of AI errors and hallucinations, and guiding students' use of AI in a way that fosters genuine learning. Preliminary results on the use of the tool are encouraging and show the effectiveness of the overall proposed methodology.

Related Work. Error analysis as a pedagogical technique is not new, even in the SQL context; e.g., Taipalus et al. [18] and Miedema et al. [12] explore students' reasoning to reveal typical errors and their causes. SQL query errors range from simple syntax issues to deeper semantic and logical misunderstandings and different classifications have been proposed over the years [2, 4, 14, 15]. Our framework adopts the taxonomy by Taipalus et al. [19], which to the best of our knowledge is the most complete and frequently used in the literature. It classifies errors as syntactic, semantic (incorrect regardless of the request), logical (correct but mismatched), or overly complex. Using this framework, we categorized student mistakes from our database courses at the University of Genoa [10, 13].

SQL learning benefits from error-aware learning tools [8], whose potential can only be leveraged if they extend beyond grading efficiency by also providing tutoring capabilities to the students. Hint generation is, e.g., the focus of [7, 9]. These approaches are at the basis of the error categorization in our framework.

Generative AI integrated in programming environments for educational use is more and more frequently exploited. In the SQL context, their effects on student performance have been analyzed in [16]. Other uses of generative AI in SQL learning are for the automatic exercise generation [1] and for the assessment of authentic learning [17]. The approach we propose is more similar but complementary to the ones proposed in [3, 11] to fine-tune GPT models to detect and provide feedback on errors in SQL queries.

Outline of the Paper. The remainder of the paper is organized as follows. Section 2 presents the framework, discusses the main methodologies we plan to follow for each framework component, and provides details about the currently operational components. Preliminary results are presented in Section 3. Finally, Section 4 concludes the paper and outlines future work.

2 AI-Powered Tutoring Framework

The goal of our framework is to enhance SQL learning by focusing on systematic error detection, AI-assisted feedback, and assignment generation. The framework relies on a multi-faceted strategy: we collect and analyze student queries, classify errors automatically, provide AI-driven assistance, and generate targeted assignments to reinforce learning.

Figure 1 illustrates the framework components. Users interact with the system through a web browser, by which they can submit queries and get the corresponding results. The framework relies on two separate database instances: `DB Users` contains a single database for each student, where students can exe-

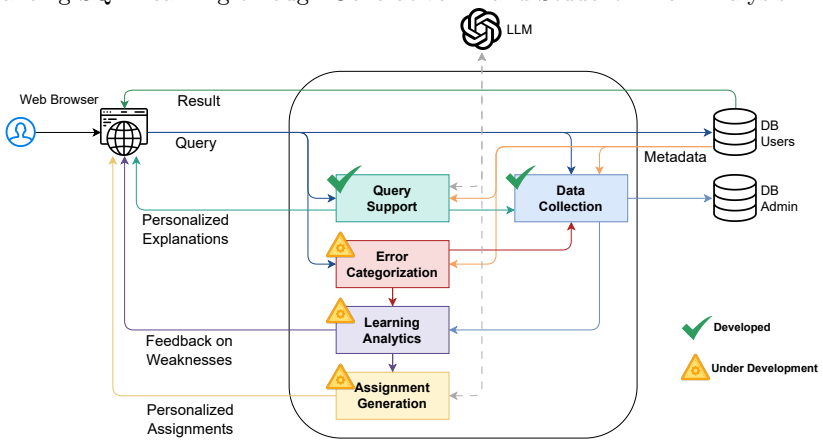

Fig. 1. Architecture of the AI-Powered Tutoring Framework

cute any operation; `DB Admin` is used only to log student queries and perform subsequent analysis. The framework is composed of five modules, described in the following: two have already been implemented (green tick in the figure) and three are currently under development (orange tick).

2.1 Data Collection Module

The Data Collection Module logs all queries executed by students along with their execution outcomes. For each query, the system records: the query text; the execution status (successful or erroneous); the timestamp; the student who submitted it; the corresponding request in natural language; the expected correct SQL solution (provided by the instructor and hidden from students), if available; the active `search_path` at the time of execution; the type of SQL statement (e.g., `SELECT`, `INSERT`, `CREATE`, etc.); the entire database structure at the time of execution (i.e. available tables, their structure, and their constraints). If the query executes successfully, the resulting output is also stored. In cases where execution fails, the associated error message is logged.

2.2 Query Support Module

The Query Support Module presents users with interactive buttons, providing tailored and meaningful explanations, depending on the query written by the user, its output, and additional metadata (e.g., the database structure). Depending on the query execution status, different options are available.

For *syntactically incorrect queries*, the module provides the following options:

- *Explain Error:* clarifies the DBMS error message in relation to the specific mistake in the query.
- *Provide an Example:* presents a simplified query that produces the same type of error, helping students understand the issue in a broader context.

- *Locate Error:* highlights the specific part of the code responsible for the error, assisting students who have not yet identified the exact cause.
- *Suggest Fix:* offers a possible correction to resolve the error.

These options are progressively made available, to encourage active problem-solving and deeper engagement with error analysis. Initially, students can access explanations and illustrative examples to help them understand the nature of the error. If they still struggle to identify the issue, they can use the error location feature to pinpoint the problematic part of their query. The suggest fix option becomes available only after students have exhausted all other support mechanisms, ensuring that they make a genuine effort to diagnose and understand the error before receiving a direct solution.

For *syntactically correct queries*, the module provides the following options:

- *Describe My Query:* generates a brief natural language summary of the user query, helping to identify potential logical or reasoning errors. If the description does not align with the user intended goal, this may indicate a misunderstanding of the query actual behavior.
- *Explain My Query:* provides a detailed step-by-step breakdown of each clause in natural language, clarifying the function of different query components. This allows students to reflect on the role of each clause and detect possible logical inconsistencies or misinterpretations.

When the user selects one button, a prompt is automatically generated and sent to the ChatGPT API (model `gpt-4o-mini`). The prompt contains only the information necessary to fulfill the selected request. For instance, the *Explain Error* button provides the model with the error message and the original query, and explicitly instructs it to explain the error without offering a correction. Additionally, a response template is embedded in the prompt to maintain stylistic consistency across outputs. To avoid misleading the model, any comment present in the SQL code is removed, as inconsistencies between code and comments can distort the model's interpretation. Figure 2 shows an example of a personalized error explanation for a query returning multiple rows instead of a single value.

These interactive features support students in refining their understanding of SQL query logic. By offering progressively detailed explanations and comparisons, we encourage self-reflection, helping students diagnose errors independently before resorting to external assistance. The proposed features create an interactive dialogue, helping students understand their mistakes and learn from them in real time. The goal is to mimic a human tutor guidance by offering just-in-time explanations and suggestions as students work through SQL problems.

While the integration of AI-driven support can enhance scalability and personalization opportunities, it is important to acknowledge potential risks. The explanations and suggestions generated by the LLM may occasionally contain inaccuracies or misinterpret the user's intent, especially in edge cases or ambiguous scenarios. To mitigate this, the system encourages critical engagement by presenting AI-generated feedback only after students have attempted to understand the request or resolve the issue themselves.

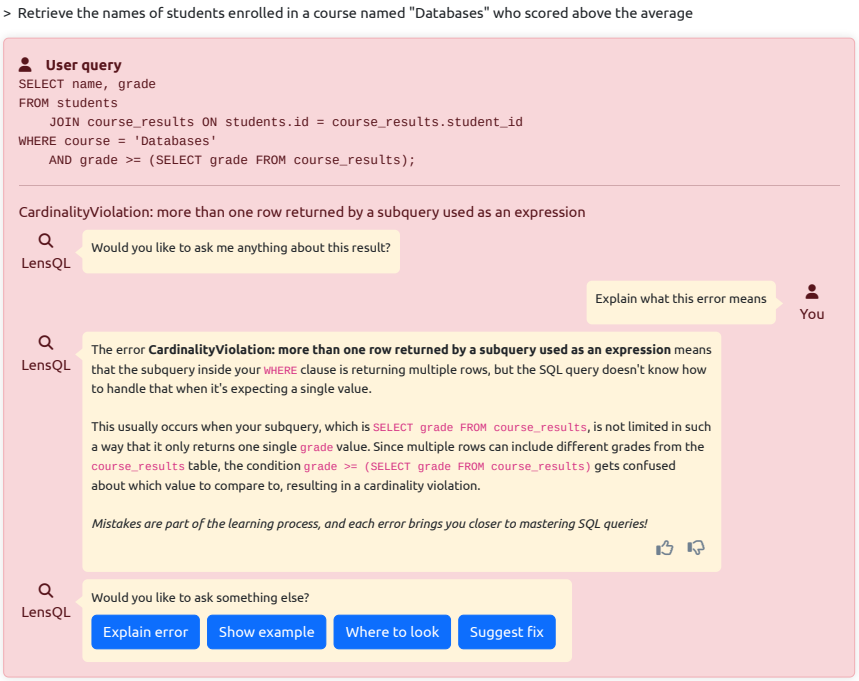

Fig. 2. Example of personalized error explanation for a syntactically-wrong query.

2.3 Error Categorization Module

To systematically address mistakes, submitted queries will be processed through an automated error detection module, currently under development. This component will classify errors based on the taxonomy proposed in [19] which delineates errors into four primary categories: syntax errors, semantic errors, logical errors, and complications. Furthermore, this taxonomy encompasses a three-level hierarchy with 105 specific error types, enabling a nuanced analysis of student mistakes. *Syntax errors* cover issues that violate SQL grammar (and thus are caught by the database engine), whereas semantic and logical errors involve queries that execute but return incorrect results. *Semantic errors* are characterized by inherent contradictions or tautologies within the query logic, rendering them incorrect regardless of the specific data demand. *Logical errors*, on the other hand, are context-dependent and arise when a query, though syntactically and semantically correct, fails to retrieve the intended data due to misalignment with the user's specific information need. *Complications* refer to unnecessary or overly complex parts of a query that do not change the result but make the solution harder to understand.

We plan to detect a substantial number of errors through parsing the SQL query structure and running it against a set of rules [4], optionally supplemented by additional metadata, such as the active `search_path` or information about

the available schema(s) at the time of execution. For more complex errors, particularly logical errors, it will also be possible to leverage the expected solution provided by the instructor [7], as well as the request in natural language.

As an example, consider the following query, which attempts to retrieve the names of students enrolled in a course named *Databases* who scored above the average:

```
SELECT name FROM students
JOIN course_results ON students.id = course_results.student_id
WHERE course LIKE Databases OR grade > AVG(grade);
```

These are the error categories the module should be able to detect:

- **Syntax Error (Undefined Database Object): omitting quotes around character data** — The string literal `Databases` is not enclosed in single quotes. As a result, it is interpreted as an identifier, leading to a syntax error if no such column or variable exists.
- **Syntax Error (Illegal Aggregate Function Placement): using aggregate function outside SELECT or HAVING** — The aggregate function `AVG(grade)` is used directly in the `WHERE` clause, which violates SQL syntax rules. Aggregate functions must appear in `SELECT`, `HAVING`, or in a scalar subquery.
- **Logical Error (Operator Error): OR instead of AND** – The use of `OR` instead of the intended `AND` changes the logic of the query, possibly returning unintended results and failing to meet the user's information need.
- **Complication: unqualified column references** — The columns `name`, `course`, and `grade` are referenced without table aliases. While syntactically valid, this reduces clarity and increases the risk of ambiguity in multi-table queries.
- **Complication: LIKE without wildcards** – The `LIKE` operator is used without wildcards, which reduces it to a simple equality check. This adds unnecessary complexity and reduces clarity.

By automatically tagging each query with these error categories, we will ensure consistency in how mistakes are identified. This automation makes it easier to spot recurring misconceptions. While the example shown involves moderate complexity, the module is being designed to eventually support more advanced queries, including those with subqueries or grouping conditions.

2.4 Learning Analytics Module

The aim of the Learning Analytics Module is to leverage error classifications and student interaction data to construct detailed learner profiles and support data-driven adaptation of instruction. For each student, the module will be able to continuously track error trends, such as the frequency and types of mistakes (syntax, semantic, or logical) across different exercises. It will also record how long students take and how many attempts they need to resolve each issue, providing

information on the learning process and persistence. Based on successful query completions and error patterns, the system will infer concepts' mastery levels across core SQL constructs such as joins, aggregation, and subqueries. Additionally, by analyzing behaviors like repeated queries and use of support features, it will estimate confidence levels, helping to identify cases of overconfidence or uncertainty.

We plan to create interactive dashboards that will provide real-time insights for both students and instructors. Students will have access to a personalized dashboard showing their progress across topics, a heatmap of error types, and a "confidence vs. accuracy" plot to reflect on how well their perceived understanding aligns with actual performance. The system will also track success streaks, defined as sequences of exercises correctly solved on the first attempt, which signal growing proficiency and can boost motivation. Instructors, on the other hand, will be presented with aggregated class-wide analytics that highlight common misconceptions, detect at-risk or high-performing students, and compare cohort performance.

To further enhance diagnostic capabilities, we are also planning to cluster students based on shared error patterns and learning behaviors, allowing instructors to tailor interventions more effectively. This will also surface high-impact misconceptions that are strongly correlated with low performance or slow correction times. Finally, by integrating engagement metrics (e.g., time on task, help interactions) and mapping student performance against predefined learning objectives, the module will produce individualized "SQL skill coverage maps", offering a comprehensive overview of student understanding and instructional needs.

2.5 Assignment Generation Module

Building on the metrics computed by the learning analytics module, this component will allow users to generate personalized SQL assignments that directly target the specific misconceptions identified in each student's queries. To do this effectively, we plan to incorporate metadata from the database, such as table names, schema structure, and attribute types, ensuring that generated exercises are both realistic and pedagogically aligned with the course content.

Using generative AI, the module will be able to produce customized exercises that mirror the context in which the student made the error, but with slight variations designed to challenge and correct the misunderstanding. For example, if a student frequently omits a `GROUP BY` clause when using aggregate functions, the system will generate a new query task requiring correct grouping to produce the intended results. This integration of prior error data and schema-aware content creation will ensure each student receives assignments that are both contextually grounded and instructionally relevant. Each assignment will consist of: (*i*) a relational schema and synthetic dataset generated to match the selected domain; (*ii*) a natural language description of the task to be solved; (*iii*) one or more expected outputs, against which students can compare their query results.

In addition to targeting specific errors, the module will also allow students to choose a topic of personal interest, which is used to generate a synthetic dataset and schema relevant to that context (e.g., sports, music, or books). By grounding exercises in topics that resonate with students' interests, we aim to enhance motivation and engagement. These assignments could be used to complement standard course exercises by offering focused practice that adapts to each student's needs, while also promoting self-paced exploration of SQL topics in personally meaningful contexts.

3 Preliminary Results

The current system was employed in the context of an *"Introduction to Databases"* second year Bachelor course [6], with 115 students. The learning outcomes of the course, like those of typical introductory database courses, are quite broad, covering a wide range of skills and abilities, that for querying includes *expressing queries and manipulating operations in the SQL language.*

To date, we have recorded a total of 82,813 student-submitted SQL queries. Approximately 60% of the queries executed successfully, while the remaining 40% contained at least one error. Notably, only 4.14% of the queries were submitted during scheduled lab sessions, while the vast majority (over 95%) were submitted outside official lab hours. This usage pattern suggests that the tool is actively supporting students beyond the constraints of in-person instruction.

Focusing on `SELECT` queries, which represent the most common command type (about 45% of the executed commands) around 70% were executed without errors. The remaining 30% included one or more errors, reflecting misconceptions and challenges related to syntax and query logic, even in basic scenarios. The most frequent error categories identified by PostgreSQL are the following[1]:

- **Undefined (47.18%)** — Situations in which something referenced in the SQL query is not defined or does not exist. For example, the student might be trying to select from a table that has not been created or spelled correctly.
- **Syntax Errors (29.28%)** — Violations of SQL grammar rules, such as missing keywords or incorrect clause ordering.
- **Grouping Errors (10.55%)** — Misuses of aggregate functions or missing `GROUP BY` clauses, one of the most conceptually challenging aspects of SQL.

Other less frequent categories include ambiguous column references due to unqualified names in multi-table queries (3.21%), data type mismatches such as comparing numeric values with strings (2.77%), invalid operations due, e.g., to datetime format or casting (2.68%), cardinality violations due to subqueries used in expressions but returning more than one row (1.56%), insufficient privileges (1.07%), and other datetime-related issues (0.49%). These results confirm prior findings that conceptual mastery of grouping operations and schema navigation remain key learning challenges for SQL novices.

[1] Note that here we are referring to the error messages raised by PostgreSQL that will be mapped to error types in the *syntactic* category of the reference taxonomy [19].

In terms of AI-assisted support, students submitted a total of 2,784 help requests through the Query Support Module. The most commonly used features were *Suggest Fix* (49% of requests) and *Explain Error* (33% of requests), showing a clear preference for actionable guidance and concise feedback. Temporal analysis of tool usage reveals distinct patterns. Query activity increased steadily over the semester, with noticeable spikes coinciding with more complex lab sessions and immediately prior to exam dates. Help requests mirrored this trend, with *Suggest Fix* and *Explain Error* being particularly prominent during peak periods. A sharp increase in student engagement was observed during weeks dedicated to advanced topics, especially for operational SQL, as well as near assessment deadlines.

4 Conclusions & Future Work

In this paper we presented a data-driven framework designed to support SQL learning by leveraging student errors as opportunities for reflection and conceptual growth. The system combines query logging, AI-generated feedback, error categorization, learning analytics, and personalized assignments to promote a deeper understanding of SQL. Rather than offering direct solutions, it fosters active reasoning and iterative learning, aligning with pedagogical principles that treat errors as a core part of the learning process. Two components — data collection and query support — have been implemented, and preliminary results show the framework is capable of identifying meaningful error patterns and is actively used as a learning aid.

Future developments will focus on completing the framework, by implementing the missing and incomplete components, and on its validation. We plan to assess the effectiveness of our approach through a comparative study using historical data from different editions of our introductory database course [6]. We will examine exam performance, both quantitatively, through grade distributions, and qualitatively, through an analysis of recurring error types, to determine whether the system contributes to more robust learning outcomes. The data collection will also enable a detailed analysis of learning trajectories over time. By analyzing how students move from incorrect to correct solutions, and how they respond to feedback and guidance, we aim to identify which features of the system are most effective in supporting progress. These insights will inform iterative refinements to the framework and lay the groundwork for broader applications. Longer-term directions include the application of the framework to different courses and learning contexts, such as K-12, and the application of the same approach to other areas within computer science education.

References

1. W. Aerts, G. Fletcher, and D. Miedema. A Feasibility Study on Automated SQL Exercise Generation with ChatGPT-3.5. In *Proc. 3rd Int'l Workshop on Data Systems Education*, pages 13–19, 2024.

2. A. Ahadi, J. Prior, V. Behbood, and R. Lister. Students' semantic mistakes in writing seven different types of SQL queries. In *Proc. ACM Conf. on Innovation and Technology in Computer Science Education*, pages 272–277, 2016.
3. A. AlRabah, S. Yang, and A. Alawini. Optimizing Database Query Learning: A Generative AI Approach for Semantic Error Feedback. In *Proc. ASEE Annual Conference and Exposition*, 2024.
4. S. Brass and C. Goldberg. Semantic errors in SQL queries: A quite complete list. *Journal of Systems and Software*, 79(5):630–644, 2006.
5. L. Cagliero, L. De Russis, L. Farinetti, and T. Montanaro. Improving the effectiveness of SQL learning practice: a data-driven approach. In *IEEE COMPSAC 2018*, volume 1, pages 980–989, 2018.
6. B. Catania, G. Guerrini, and D. Traversaro. Collaborative Learning in an Introductory Database Course: A Study with Think-Pair-Share and Team Peer Review. In *Proc. 1st Int'l. Workshop on Data Systems Education*, page 60–66, 2022.
7. Y. Hu, A. Gilad, K. Stephens-Martinez, S. Roy, and J. Yang. Qr-hint: Actionable hints towards correcting wrong sql queries. *Proc. ACM on Management of Data*, 2(3):1–27, 2024.
8. C. Kenny and C. Pahl. Automated tutoring for a database skills training environment. In *Proc. 36th SIGCSE Technical Symposium on Computer Science Education*, pages 58–62, 2005.
9. C. Kleiner and F. Heine. Enhancing Feedback Generation for Autograded SQL Statements to Improve Student Learning. In *Proc. Conf. on Innovation and Technology in Computer Science Education*, page 248–254, 2024.
10. A. Livani. Do the errors produced by generative AI in formulating queries reflect students' misconceptions in learning SQL? Master's thesis, MSc in Computer Science, University of Genova, 2024.
11. K. Manikani, R. Chapaneri, D. Shetty, and D. Shah. SQL Autograder: Web-based LLM-powered Autograder for Assessment of SQL Queries. *Int. Journal of Artificial Intelligence in Education*, pages 1–31, 2025.
12. D. Miedema, E. Aivaloglou, and G. Fletcher. Identifying SQL Misconceptions of Novices: Findings from a Think-Aloud Study. In *Proc. ACM Conf. on International Computing Education Research*, page 355–367, 2021.
13. D. Ponzini, A. Livani, G. Guerrini, B. Catania, and M. Coccoli. Analyzing Common Student Errors in SQL Query Formulation to Enhance Learning Support. In *Proc. 33rd Symposium on Advanced Database Systems*, 2025.
14. S. Poulsen, L. Butler, A. Alawini, and G. L. Herman. Insights from Student Solutions to SQL Homework Problems. In *Proc. ACM Conf. on Innovation and Technology in Computer Science Education*, pages 404–410, 2020.
15. K. Presler-Marshall, S. Heckman, and K. Stolee. SQLRepair: Identifying and Repairing Mistakes in Student-authored SQL Queries. In *Proc. IEEE/ACM ICSE-SEET*, pages 199–210, 2021.
16. C. Ramakrishnan, S. Cassidy, and M. Bower. Evaluating Student Performance and Interactions in Generative AI-Integrated SQL Practical Tests. In *Proc. ACM Conf. on Innovation and Technology in Computer Science Education*, 2025.
17. R. Sooriamurthi, X. Tu, and A. E. C. Pensky. A Generative AI Tool to Foster and Assess Authentic Learning: A Case Study in Teaching SQL. In *Proc. ACM Conf. on Innovation and Technology in Computer Science Education*, page 465–471, 2025.
18. T. Taipalus. Explaining Causes behind SQL Query Formulation Errors. In *Proc. IEEE Frontiers in Education Conference*, pages 1–9, 2020.
19. T. Taipalus, M. Siponen, and T. Vartiainen. Errors and Complications in SQL Query Formulation. *ACM Transactions on Computing Education*, 18(3):1–29, 2018.