

SQL JUDGE: DESIGN AND IMPLEMENTATION OF AN AUTOMATED SQL QUERY GRADING PLATFORM FOR HIGH SCHOOL INFORMATICS EDUCATION

Nguyen Huu Bao, Le Xuan Hien* and Pham Tho Hoan

Hanoi National University of Education, Hanoi, Vietnam

*Corresponding author: Le Xuan Hien, e-mail: lexuanhien@hnue.edu.vn

Received March 27, 2026. Revised July 17, 2026. Accepted September 30, 2026.

Abstract. This paper presents Structured Query Language (SQL) Judge, a web-based automated SQL grading platform designed for high school informatics education. The study follows a design-and-evaluation methodology consisting of problem analysis, system design, implementation, and pilot evaluation. The main contribution is not a new theory of SQL equivalence, but an integrated engineering solution that combines result-based grading, multi-layer sandboxing, workspace isolation, and teacher-oriented management with configurable SQL-construct requirements in a lightweight platform suitable for school-level deployment. Initial experiments were conducted in a controlled local school-network pilot deployment involving 62 users, 43 SQL questions, and 486 student submissions. The results show that SQL Judge successfully supported query submission, controlled execution, result-based grading, and immediate feedback delivery. The system maintained an average end-to-end response time of 1.42 seconds per submission, with 95% of submissions processed within 4.7 seconds. For complex queries, the average response time was 2.83 seconds, with 95% completed within 6.5 seconds. In addition, the sandbox mechanism blocked all 36 deliberately designed security test cases involving destructive commands, suspicious SQL patterns, and resource-intensive queries. These findings suggest that SQL Judge is technically feasible as a safe and responsive environment for SQL practice and formative assessment; however, larger-scale studies are still needed to validate grading accuracy, workload reduction, scalability, and educational effectiveness.

Keywords: SQL injection, security sandbox, automated grading, SQL Judge, result-based SQL assessment, workspace isolation, research methodology, and IT education.

1. Introduction

In the high school informatics curriculum, understanding relational database operations is a fundamental prerequisite for cultivating students' computational thinking capabilities. Structured Query Language (SQL) serves as the ubiquitous standard for

interacting with relational database management systems (RDBMS), enabling users to retrieve, filter, and analyze structured data. Through formal SQL instruction, students gain practical experience in data management while fostering cognitive and analytical skills essential for structured problem-solving. Despite its pedagogical significance, teaching and evaluating SQL proficiency in secondary education presents persistent didactic and technological challenges [1].

One of the primary difficulties arises from the non-procedural, declarative nature of SQL. Unlike procedural programming languages, SQL allows diverse query logic to yield semantically identical result sets. For instance, equivalent data outputs can be achieved using JOIN operations, nested subqueries, or Common Table Expressions (CTEs). This syntactic variability renders conventional grading methodologies - such as direct string or pattern matching- inherently ineffective. Consequently, educators must manually inspect and evaluate student submissions, a process that is labour-intensive, susceptible to subjective bias, and prone to delaying informative feedback crucial to effective learning.

While numerous automated assessment systems (Online Judges) - such as LeetCode, HackerRank, or various competitive programming platforms – are widely deployed, these tools are primarily engineered for professional developers or undergraduate Computer Science curricula, exhibiting steep technical complexity. In the context of high school informatics education, students require a more intuitive interface capable of translating runtime execution errors into pedagogically meaningful feedback rather than raw system tracebacks. Furthermore, secondary education requires test suites that prioritize fundamental relational concepts and core curriculum competencies over advanced algorithmic optimization.

Another critical challenge pertains to establishing a secure execution environment for SQL practice. Executing unrestricted, user-submitted SQL statements on a server introduces severe security vulnerabilities, including resource-intensive queries that trigger Denial-of-Service (DoS) conditions, SQL injection attempts, or malicious schema alterations via destructive commands such as DROP or ALTER [2], [3]. Compared to enterprise solutions that demand complex complex infrastructure, SQL Judge strives for architectural efficiency by optimizing sandbox security at the application layer, ensuring robust functionality within the constrained IT environments typical of high schools.

To address these limitations, this paper presents SQL Judge, an automated assessment platform specifically tailored to facilitate SQL instruction within high school informatics education. The proposed system establishes a lightweight, isolated execution sandbox powered by Role-Based Access Control (RBAC), and an automated grading engine evaluates student submissions by comparing query outputs against predefined datasets and test cases. By enabling secure query execution and immediate feedback, the platform effectively reduces teacher assessment workloads while enhancing student learning outcomes in practical SQL applications.

2. System architecture and design

Automated assessment has been extensively investigated in computer science education as an effective mechanism to alleviate instructional workload while delivering rapid formative feedback. In programming courses, online judge platforms typically

evaluate source code by executing submissions against standardized, predefined test suites. However, automated SQL assessment fundamentally diverges from conventional programming assessment because SQL is declarative: several syntactically different queries may be semantically equivalent and produce the same result. Therefore, direct string comparison is not reliable for grading SQL submissions [4], [5].

Early SQL assessment systems commonly used result-based output comparison, where a student query is executed on a prepared database, and its output is compared with the expected result. This approach is simple and practical, especially for introductory database courses. However, purely binary grading may not provide sufficient diagnostic feedback because a query is marked either correct or incorrect. Later studies introduced partial grading methods by considering syntax, schema, result correctness, query structure, or edit distance between student submissions and reference solutions. These methods improve fairness and help students understand their mistakes, but they often require more complex implementation and careful configuration by instructors [6], [5].

Intelligent tutoring systems for SQL have also been developed to support personalized feedback. For example, recent work on ItsSQL emphasizes guided SQL learning and individualized feedback while reducing the effort required from instructors[7]. Other recent studies have investigated sensitivity in automated SQL grading, graph-based semantic similarity, and neural-network-based grading methods [6], [8], [9]. These approaches indicate that SQL grading is moving beyond simple result comparison toward more fine-grained analysis of student submissions.

Compared with these approaches, SQL Judge focuses on a different educational context. The system is not intended to replace advanced SQL tutoring systems or propose a new theoretical model for semantic query equivalence. Instead, it aims to provide a practical and secure automated grading platform for high school informatics education. Its design emphasizes simple deployment, safe execution, immediate feedback, and teacher-oriented management. The combination of result-based grading, multi-layer sandboxing, and workspace isolation is therefore positioned as an integrated engineering solution for secondary school SQL learning rather than as a purely theoretical grading model.

2.1. Three-Tier architecture and data model

The SQL Judge platform adopts a distributed three-tier architecture to ensure scalability and maintainability while maintaining a clear separation between business logic and user interface components [10].

The presentation tier is implemented using the React library [11]. This layer is responsible for user interface rendering, application state management, and routing. It provides an interactive SQL editor that allows students to write and execute queries, while instructors access an administrative dashboard for managing exams, questions, and student submissions.

The application tier is developed using the Python-based Flask framework and serves as the central processing component of the platform. This layer exposes RESTful APIs, manages user authentication through JSON Web Tokens (JWT), coordinates SQL query execution, and performs automated grading by comparing query outputs with predefined test cases.

The data tier provides persistent storage and enables interaction with the underlying relational database management system. SQLAlchemy Object-Relational Mapping (ORM) technology [12] is used to facilitate secure and flexible database operations with supported systems such as SQLite and MySQL.

The data model is designed based on the SQL learning and assessment workflow. The Entity–Relationship Diagram (ERD) consists of five core entities: Users, Exams, Questions, Submissions, and Test Cases. Among these, the Submissions table functions as a central relational component that stores students’ SQL queries together with evaluation metadata, including execution time, memory usage, and corresponding test-case matching results.

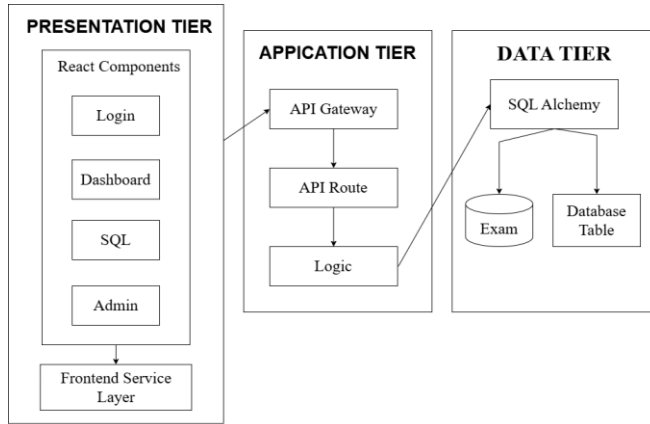


Figure 1. The 3-Tier Model Design

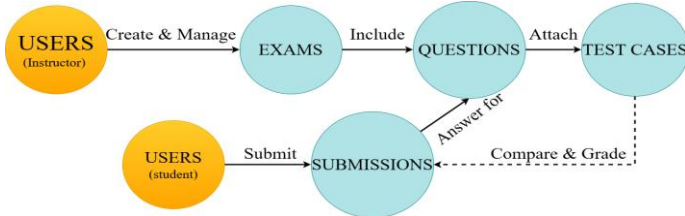


Figure 2. Data model and grading relationships among users, exams, questions, submissions, and test cases

2.2. Core technical solution of the multi-layer sandbox shield

Executing user-submitted SQL queries on a server introduces significant security risks. Since trusted code cannot be assumed secure by default, a proactive defense mechanism is required to ensure safe execution [2], [13]. To address this challenge, the SQL Judge platform implements a pipeline-based multi-layer sandbox mechanism that inspects and controls queries before they reach the underlying database. This mechanism operates as a multi-stage security pipeline.

The first protection layer performs operation validation through lexical analysis. Incoming SQL commands are examined at the token level to determine whether they belong to the set of operations permitted for the specific assignment. For example, in exercises that focus solely on data retrieval, modification commands such as INSERT, UPDATE, or DELETE are automatically rejected. This whitelist-based approach ensures that students interact with the database only within the intended scope of the task [14].

The second layer applies dangerous keyword detection to prevent structural damage to the database. A lightweight filtering mechanism identifies destructive schema-related commands such as DROP, ALTER, and TRUNCATE. Queries containing these keywords are immediately blocked before execution, thereby protecting the database schema from unauthorized alterations.

The third layer focuses on SQL injection pattern detection. To mitigate common logic manipulation attacks, regular expression matching is used to identify suspicious query patterns such as comment-based bypasses (--, /*), union-based extraction (UNION SELECT), and logical manipulation (OR 1=1) [2]. Queries matching these patterns are rejected to preserve the integrity of the execution environment.

At the final stage, the sandbox environment enforces strict resource management policies during query execution. Each query is executed under constraints on runtime and result size. A timeout mechanism prevents excessively long executions, while the result set is truncated if it exceeds a safe threshold. These safeguards prevent server overload caused by inefficient queries or excessive output volume [3].

Through this multi-layered design, the sandbox mechanism ensures that SQL queries submitted by students can be executed safely while preserving system integrity and high availability.

2.3. Automated grading engine and workspace isolation

The SQL Judge platform evaluates student submissions using an automated grading engine based on dynamic result matching techniques commonly used in automated SQL grading systems, rather than relying on static source-code analysis. In this approach, the correctness of a SQL query is determined by comparing its execution output with the expected results defined in the problem test cases [4]. Each student query is executed on a predefined sample database, and the generated result set serves as the basis for evaluation.

The grading process consists of two main comparison stages. The first stage performs structural comparison, in which the returned result set is validated against the expected schema. This step verifies the total number of columns, their specific names, and their exact order in the output table. The second stage focuses on data content comparison, where the returned values are examined row-by-row to determine whether they match the expected outputs. Special attention is given to the handling of NULL values in order to ensure consistent comparison results when different query implementations produce logically identical outputs.

In addition to output-based comparison, SQL Judge also supports instructor-defined SQL constraint checking. When creating a question, instructors can optionally specify required SQL constructs that must appear in a valid student submission, such as JOIN, GROUP BY, HAVING, ORDER BY, aggregate functions, or nested SELECT statements. This feature is useful in educational contexts because some exercises are designed not only to obtain a correct result, but also to require students to practice a specific SQL concept.

During grading, the system first checks whether the submitted query satisfies the required structural constraints defined by the instructor. If a required construct is missing, the system can return targeted feedback indicating which SQL element should be

included. If no such constraint is configured, the system proceeds with ordinary result-based grading. This design preserves the flexibility of result-based assessment while allowing instructors to align each exercise with specific learning objectives.

For example, if the purpose of an exercise is to assess students' ability to combine data from multiple tables, the instructor may require the use of a JOIN operation. In this case, a query that accidentally produces the correct output without using JOIN may still be marked as incomplete or pedagogically insufficient. Therefore, instructor-defined constraint checking helps SQL Judge evaluate not only output correctness but also the intended SQL skill targeted by the exercise.

A technical challenge arises when multiple students execute SQL queries concurrently, particularly when the queries involve Data Manipulation Language (DML) operations that modify the underlying dataset. In such cases, submissions that alter the database state may cause data race conditions, which can interfere with the evaluation results of other submissions and lead to inconsistent grading outcomes.

To resolve this problem, the proposed architecture adopts a multi-database workspace strategy. The platform automatically provides an isolated database instance from the master database to create isolated workspaces for each student session. Each workspace functions as an independent execution environment, ensuring that queries executed by one student do not interfere with the data used by others.

Through this workspace isolation mechanism, the system maintains data consistency and enables reliable automated grading even when a large number of students submit queries simultaneously.

3. Experimental setup and evaluation methodology

To enhance the reproducibility of the evaluation, the experimental procedure reports not only the number of users, questions, and submissions, but also the technical environment in which SQL Judge was deployed. Since the current experiment was conducted in a controlled local school-network pilot environment, the results should be interpreted as a classroom-scale pilot evaluation rather than a large-scale performance benchmark. The server configuration, operating system, database management system, software stack, and network settings are summarized in Table 1.

Table 1. Experimental environment for reproducibility

Component	Configuration
Server CPU	13th Gen Intel Core i7-13650HX, 14 cores
Memory	16 GB RAM
Storage	NVMe SSD
Database management system	SQLite 3.x as the default embedded database; optional support for MySQL 8.2.0
Operating system	Microsoft Windows 11 Home, 64-bit
Backend stack	Python 3.14.0, Flask 3.0.0, Flask-SQLAlchemy 3.1.1, SQLAlchemy 2.0.23
Frontend stack	React-based web interface, TypeScript, Vite, Tailwind CSS
Network setting	Controlled local school-network pilot deployment

The experiment was conducted in a controlled local school-network pilot environment. This setting was sufficient for evaluating the functional feasibility, grading workflow, response behavior, and security mechanisms of SQL Judge at a classroom scale. However, because the system was not deployed on a dedicated production server, the results should not be interpreted as evidence of large-scale system scalability. Future evaluations should involve deployment on an institutional server or cloud-based environment with a larger number of concurrent users.

The pilot involved 62 users, 43 SQL questions, and 486 student submissions collected during classroom-scale empirical testing. The questions covered basic SQL operations, filtering, sorting, aggregation, joins, nested queries, and simple data definition tasks. The system successfully supported the complete workflow of SQL practice, including query submission, controlled execution, result-based grading, and immediate feedback delivery. The indicators proposed for future grading-accuracy evaluation are summarized in Table 2.

Table 2. Proposed grading-accuracy indicators for future large-scale evaluation

Indicator	Meaning	Suggested measurement	Current status
Agreement with human grading	The proportion of submissions for which SQL Judge and instructor judgment are identical	Number of matched grading decisions / total reviewed submissions	Planned for future large-scale evaluation
False positive rate	Incorrect submissions accepted by the system	False positives / total incorrect submissions	Not yet measured
False negative rate	Correct or semantically acceptable submissions rejected by the system	False negatives / total correct submissions	Not yet measured
Baseline comparison	Comparison with a simple string-matching grader or manual grading time	Accuracy, processing time, and feedback delay	Planned
System response time	Time required to evaluate a submission and return feedback	Mean and maximum response time across submissions	Observed in pilot testing
Workload reduction	Reduction in instructor grading time compared with manual assessment	Manual grading time compared with automated grading time	Planned

The indicators in Table 2 are proposed to guide future large-scale evaluations of SQL Judge. In the present pilot stage, a complete comparison with human grading has not yet been conducted. Therefore, the current evaluation focuses mainly on system feasibility, response behavior, and security handling. A full grading-accuracy evaluation will require each student submission to be independently reviewed by an instructor and then compared with the grading decision generated by SQL Judge.

This evaluation design is important because SQL exercises can often be solved using different but semantically equivalent query formulations. For this reason, simple string matching is insufficient for reliable SQL assessment. A result-based grading approach is more appropriate because it evaluates the returned schema and data values rather than the exact textual form of the submitted query. In future experiments, SQL Judge should therefore be evaluated against both human grading and a simple string-matching baseline to measure inter-rater grading agreement, false positives, false negatives, response time, and instructional workload reduction.

4. Results and discussion

The experimental results demonstrate that SQL Judge performed reliably in terms of execution stability, response behavior, security handling, and pedagogical support. The pilot performance results are summarized in Table 3

Table 3. Pilot performance results of SQL Judge

Metric	Result
Number of users	62
Number of SQL questions	43
Number of student submissions	486
Average end-to-end response time	1.42 seconds
Median response time	0.86 seconds
95th percentile response time	4.7 seconds
Average response time for complex queries	2.83 seconds
95th percentile response time for complex queries	6.5 seconds
Security-oriented test cases	36
Blocked unsafe SQL inputs	36 / 36
Successfully executed submissions	425
Failed or error submissions	61
Query success rate	87.4%

4.1. Stability and execution performance

During the pilot test, SQL Judge successfully executed 425 out of 486 submitted SQL queries, corresponding to a query success rate of 87.4%. The remaining 61 submissions, accounting for 12.6%, resulted in failure or error responses. Most of these unsuccessful submissions were caused by user-side syntax or semantic errors, such as misspelled table

names, missing parentheses, incorrect keywords, invalid column references, or incomplete query structures. This result indicates that the system was able to reliably execute valid SQL submissions while effectively detecting and reporting invalid student attempts during the learning process.

A small number of failed submissions were also related to queries that violated sandbox restrictions or exceeded predefined execution constraints. These cases show that the system did not simply execute all user inputs without control, but applied protective mechanisms to prevent unsafe or resource-intensive operations.

The platform maintained an average end-to-end response time of 1.42 seconds per submission, measured from query submission to feedback display. The median response time was 0.86 seconds, and 95% of submissions were processed within 4.7 seconds. For more complex queries involving joins, aggregation, or nested SELECT statements, the average response time was 2.83 seconds, with 95% completed within 6.5 seconds. These results suggest that SQL Judge is sufficiently responsive for classroom-scale SQL practice and formative assessment in a local school-network environment.



Figure 3. Query success rate

4.2. Security capability

The sandbox filtering mechanism demonstrated robust defensive capabilities during the security evaluation. In addition to ordinary student submissions, the system was tested with 36 deliberately crafted unsafe SQL inputs. These test cases included 15 SQL injection attempts, 12 destructive command attempts, and 9 resource-intensive query attempts.

The system successfully detected and blocked all 36 unsafe test cases before execution. SQL injection attempts were rejected via pattern validation and query filtering rules. Destructive commands, such as operations related to dropping, altering, deleting, or truncating database objects, were prevented by the sandbox policy. Resource-intensive queries were also throttled to mitigate the risk of excessive execution time or system overload.

These results demonstrate that the sandbox mechanism can effectively secure the underlying database from common unsafe SQL inputs in the tested scenarios. Although further security evaluations are still needed in larger and more diverse environments, the pilot results indicate that SQL Judge provides a safer environment for students to practice SQL compared with unrestricted query execution environments.

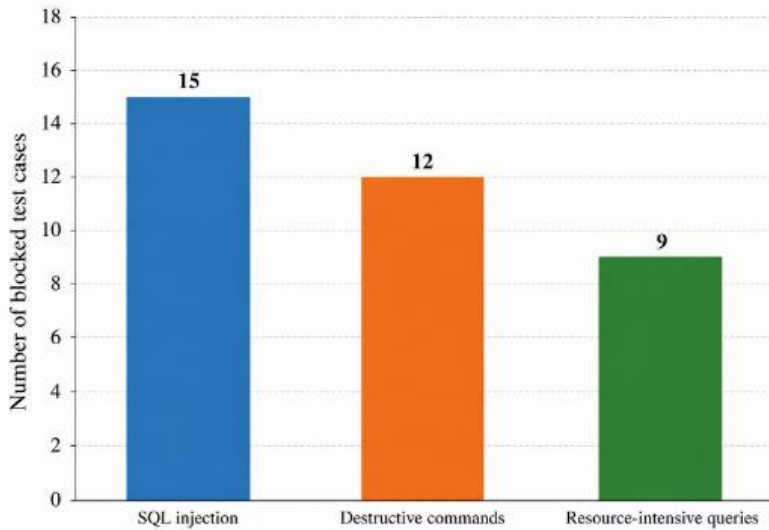


Figure 4. Security test cases detected and blocked

4.3. Pedagogical value

From a pedagogical perspective, SQL Judge supports both students and instructors in the SQL learning process. For students, the system provides immediate feedback after query submission, allowing them to identify syntax errors, semantic flaws, and mismatches between their output and the expected results. This immediate response is crucial for formative assessment because students can revise their queries and learn from their mistakes during practice.

For instructors, the platform helps reduce repetitive manual grading tasks by automatically executing submissions, comparing returned results with expected outputs, and recording submission histories. The system also provides analytical data, such as submission status, query success rate, error patterns, and performance across different question types. These metrics can help instructors identify common learning difficulties and evaluate students' practical SQL competencies more systematically.

Another pedagogical advantage of SQL Judge is that instructors can configure required SQL constructs for each question according to the intended learning objective. For example, an instructor may require students to use JOIN in exercises about combining tables, GROUP BY in aggregation tasks, or nested queries in lessons about subquery logic. This option allows the system to support concept-oriented practice rather than merely checking whether the final output is correct. As a result, SQL Judge can better align automated assessment with classroom teaching goals and help instructors verify whether students have practiced the intended SQL technique.

However, the system should not be considered a complete replacement for human judgement. While result-based grading is suitable for many structured SQL exercises, instructors remain essential for designing high-quality questions, reviewing complex or semantically alternative answers, and making final assessment decisions. Therefore, SQL Judge is best positioned as a supporting tool that improves grading efficiency, provides immediate formative feedback, and enhances the management of SQL practice activities in high school Informatics education.

Overall, the pilot results demonstrate the technical feasibility of SQL Judge for classroom-scale SQL practice and formative assessment. Nevertheless, larger-scale empirical experiments are still required to further validate grading accuracy, reduce instructional workload, scalability under concurrent access, and educational effectiveness across different teaching contexts.

5. Conclusions

This paper has detailed the design and implementation process of the SQL Judge educational platform. The research addresses several practical limitations of traditional assessment methods by resolving the execution environment isolation problem through a multi-layer sandbox architecture and developing an automated grading engine based on dynamic result matching. The platform protects system data from security threats, provisions a controlled practice environment, provides immediate feedback to students, and reduces instructional workload for instructors.

Although the initial results are promising, the system still requires further validation before large-scale adoption. In future work, the experimental evaluation will be expanded to include a larger cohort of students, higher submission volumes, and more diverse SQL question types. The evaluation will also include explicit server configuration, grading-accuracy metrics, and comparison with baseline systems such as string-matching graders and manual grading.

From a technical perspective, future development will focus on transitioning the system toward a microservice-based architecture that can be containerized and orchestrated on Docker and Kubernetes platforms. Another important direction is to incorporate machine learning or program-analysis techniques to automatically analyze the Abstract Syntax Tree (AST) of failed submissions and generate more specific feedback. This development would help transform SQL Judge from an automated grading tool into a more intelligent virtual teaching assistant for SQL education.

Note for Contributor:

- Author Bio: Nguyen Huu Bao is a researcher at Hanoi National University of Education, Hanoi, Vietnam; Le Xuan Hien is a lecturer at Hanoi National University of Education, Hanoi, Vietnam; Pham Tho Hoan is an Associate Professor at Hanoi National University of Education, Hanoi, Vietnam.

- Author's contributions: Nguyen Huu Bao: conceptualization, software implementation, writing-original draft; Le Xuan Hien: methodology, writing -review & editing; Pham Tho Hoan: software architecture, visualization, supervision.

Conflicts of Interest. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

REFERENCES

- [1] P. Ihanola, T. Ahoniemi, V. Karavirta & O. Seppälä, "Review of recent systems for automatic assessment of programming assignments", *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, pp. 86-93, 2010.

- [2] W. G. J. Halfond, J. Viegas & A. Orso, “A classification of SQL injection attacks and countermeasures”, *Proceedings of the International Symposium on Secure Software Engineering (ISSSE)*, pp. 13-22, 2006.
- [3] M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*, O’Reilly Media, 2017.
- [4] Z. Wang & et al., “Combining dynamic and static analysis for automated grading SQL statements”, *Journal of Information Hiding and Multimedia Signal Processing*, vol. 11, no. 1, pp. 179-190, 2020.
- [5] B. Chandra, et al., “Automated Grading of SQL Queries”, *IEEE publication on edit-based and partial grading of SQL submissions*, 2019.
- [6] B. G. Wanjiru, P. van Bommel, and D. Hiemstra, “Sensitivity of Automated SQL Grading in Computer Science Courses”, *Proceedings/Preprint on automated SQL grading sensitivity*, 2024.
- [7] S. A. Reid, F. Kammer, J. Kunz, T. Pellekorne, M. Siepermann, and J. Wölfer, “ItsSQL: Intelligent Tutoring System for SQL”, arXiv preprint arXiv:2311.10730, 2023.
- [8] L. Köberlein, D. Probst, and R. Lenz, “Quantifying Semantic Query Similarity for Automated Linear SQL Grading: A Graph-based Approach”, arXiv preprint arXiv:2403.14441, 2024.
- [9] D. R. Schwartz and P. Rivas, “An Automated SQL Query Grading System Using an Attention-Based Convolutional Neural Network”, 2022.
- [10] A. S. Tanenbaum and M. van Steen, *Distributed Systems*, 3rd ed. CreateSpace Independent Publishing Platform, 2017.
- [11] Meta Platforms Inc., *React Documentation (v19)*, April 7, 2026. Available: <https://react.dev>
- [12] SQLAlchemy Authors, *SQLAlchemy 2.0 Documentation*, April 7, 2026. Available: <https://docs.sqlalchemy.org>
- [13] W. G. J. Halfond and A. Orso, “AMNESIA: Analysis and monitoring for neutralizing SQL injection attacks”, *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 174-183, 2005.
- [14] A. Silberschatz, H. F. Korth & S. Sudarshan, *Database System Concepts*, 7th ed. McGraw-Hill, 2019.