API Interface for Analyzing the Correctness of DML Queries

Michal Kvet, Andrea Meleková, Dmytro Demchenko
University of Zilina
Žilina, Slovakia
Michal.Kvet@uniza.sk, melekova2@stud.uniza.sk, demchenko@stud.uniza.sk

***Abstract*—Although automated assessment tools have seen significant advancements in recent years, addressing the complexities of analyzing and correcting Data Manipulation Language commands remains a challenge. In educational settings, particularly at institutions like the Faculty of Computer Science and Management, University of Zilina, manual assessment processes can be time-consuming and prone to errors. In response to these challenges, this paper introduces a sophisticated tool aimed at streamlining the analysis and correction of DML commands within the EXAM system. This tool not only automates the tedious aspects of assessment but also offers functionalities such as API design, analysis of Oracle database functions, and integration with existing examination systems. By addressing these critical areas, the tool presents a promising solution to enhance educational assessment practices, ensuring accuracy, efficiency, and fairness in grading processes.**

I. Introduction

In the realm of educational assessment and database management, the efficient analysis and automation of Data Manipulation Language (DML) commands stand as pillars of utmost importance. This paper aims to contribute by proposing the design and implementation of a robust tool tailored to analyse DML command results and streamline the correction of student solutions within the EXAM system at the Faculty of Computer Science and Management, University of Zilina (FRI UNIZA).

Existing public systems dedicated to automated control of DML commands fall into three main categories: tools comparing the data returned by individual commands, editors comparing only the textual input, and complex libraries primarily focused on parsing and evaluating these commands. To provide a clearer overview, let's delve into each of these categories.

One of these tools compare based on textual representation of commands, exemplified by websites such as onlinetextcompare, specialize in comparing textual inputs, highlighting differences between two sets of data. They are suitable for detecting typos or errors in larger files and highlighting changes made to commands. However, they may not offer significant advantages for more complex analyses. Another type of technology involves software solutions that compare the data returned by database. Examples include Oracle SQL Developer or DbVisualizer [1]. These editors often format commands according to SQL standards and highlight discrepancies between them. While useful for identifying syntactical errors or discrepancies, they may lack the depth required for comprehensive analysis. Finally, there are libraries like JSqlParser, which provide more advanced parsing capabilities in programming languages like Java.

These libraries focus on parsing SQL commands into a hierarchical structure and offer insights into their syntactic correctness [2]. However, they may not support error correction or command transformation, limiting their utility for comprehensive DML command analysis. While these technologies provide valuable insights into the processing of DML commands, they often lack comprehensive functionality for evaluation, correction, and transformation of these commands. These technologies provide valuable insights but often lack comprehensive functionality for evaluation, correction, and transformation of these commands.

At FRI UNIZA, lecturers and educators face challenges in assessing students' proficiency in database languages, particularly within exercises demonstrating SQL language syntax and query optimization. Manual scrutiny of each student authored DML command becomes laborious, necessitating an efficient alternative.

Our project addresses this challenge by developing an application designed to compare distinct SQL statements, yielding a percentage of equality between them. Instead of executing students' statements directly, the application employs a database connection to execute predefined, hardcoded SQL commands that are deemed secure.

The proposed solution features an intricate Application Programming Interface (API) designed to process both single and dual commands, with a focus on analysing standard Oracle Database Management System (DBMS) functions and addressing errors within functions, keyword discrepancies, and transformation intricacies associated with different connection types.

Rigorous testing and meticulous analysis validate the robustness, reliability, and efficacy of the developed solution, ensuring seamless integration with the existing FRI testing system and alignment with institutional objectives.

This paper aims to offer a comprehensive framework for addressing the challenges in analysing and automating DML command processing within educational settings. By harnessing the power of database management systems and leveraging cutting-edge automation technologies, the proposed

solution seeks to redefine educational assessment paradigms and foster enhanced learning experiences.

II. Existing Solutions and Related Work Analysis

A. DML Query Analysis in Literature

While there are existing tools and research efforts addressing aspects of DML query analysis, it's important to note that some of these solutions may not be publicly available or widely documented. Additionally, certain tools might focus solely on syntactical validation or textual comparison of queries without delving into the logical structure or semantic analysis.

In academic or industry settings, proprietary solutions or internal tools developed by organizations may exist to address specific needs related to DML query analysis. However, these tools might not be publicly disclosed due to proprietary considerations or intellectual property rights.

Furthermore, some research efforts might compare only the textual aspects or final results of queries without considering the logical structure or semantic correctness. This narrow focus may overlook crucial aspects of query analysis, such as ensuring data integrity, compliance with business rules, or optimization potential.

Thus, while existing tools and research efforts contribute to the field of DML query analysis, the gaps highlighted in the literature review may persist due to various factors, including limited accessibility, narrow scope, or proprietary nature of the solutions.

B. Existing Solutions

In developing our API for Analyzing the Correctness of DML Queries, our journey began with a thorough exploration of existing solutions aimed at tackling analogous challenges encountered in educational assessment within the realm of database systems. Our quest led us to encounter two remarkable papers that significantly influenced our approach to addressing this complex problem.

The first paper, authored by Charles Boisvert, Konstantinos Domdouzis, and Joshua License, introduces a blended learning approach to teaching Relational Database Systems. Central to their module is TestSQL, a query tool designed to provide students with automated feedback on SQL query exercises. However, the authors opt not to employ TestSQL for student assessment. Instead, assessment is conducted through a diverse array of questions covering various aspects of the field. Through a survey of student attitudes and assessment data analysis, the effectiveness of this approach is evaluated. The analysis reveals that students who utilize a broader range of resources tend to achieve better results. Furthermore, the study highlights that success in different sub-topics of the course is not strongly correlated, indicating that students may excel in certain areas while struggling in others. Notably, the paper underscores the significance of indirect SQL questions as the best predictor of success in other sub-topics, suggesting the importance of broadening the assessment of SQL skills to encompass various dimensions of relational database knowledge [3].

In contrast, the second paper introduces the XDa-TA system, authored by Amol Bhangdiya, Bikash Chandra, Biplab Kar, Bharath Radhakrishnan, K. V. Maheshwara Reddy, Shetal Shah, and S. Sudarshan. This system offers a solution for the automated grading of SQL query assignments, addressing the limitations of traditional assessment methods. By leveraging the XData system, XDa-TA generates datasets specifically tailored to uncover common errors in student queries. The grading process involves comparing student query results with those of correct queries against these meticulously crafted datasets. Instructors and teaching assistants can utilize this tool to streamline the grading process, providing immediate feedback to students and potentially revolutionizing assessment methodologies, particularly in Massive Open Online Courses (MOOCs) [4].

The third paper, titled "Automated Grading of SQL Queries," further explores automated grading methodologies for SQL queries. Authored by Michael Martin and Mark Nathan, this paper introduces a system designed to automatically grade SQL queries based on various criteria, including correctness, performance, and style. The system utilizes a rule-based approach to evaluate student queries, providing feedback on syntax errors, logical errors, and performance issues. By automating the grading process, instructors can save time and provide timely feedback to students, enhancing the learning experience [5].

Lastly, the paper "Automatic, Configurable, and Partial Assessment of Student SQL Queries with Joins and Groupings," authored by Goran Đambić, Mario Fabijanić, and Ana Lokas Ćošković, presents a fully automated assessment model for SQL queries, particularly focusing on complex joins and groupings. This system aims to provide partial assessment capabilities, allowing for customizable grading rules to adapt to institutional standards. By automating the assessment process, teachers can reduce their workload, ensure consistency in grading, and provide faster feedback to students. The system compares favorably with manual assessments, even for complex queries, demonstrating its potential for application in higher education settings [6].

By examining the approaches, methodologies, strengths, weaknesses, and contributions of each system alongside insights from other recent studies or advancements, we aim to identify emerging trends, challenges, and opportunities for future research and development.

III. Solution Areas Covered

A. Approach

Our system operates under the premise that users do not require prior familiarity with database environments, SQL syntax, or the Oracle Database Management System (DBMS). Instead, it handles these complexities internally, providing a user-friendly interface that abstracts away technical intricacies. This approach ensures accessibility and ease of use for educators and administrators involved in the grading process.

Additionally, the system assumes a consistent logical syntax in the database structure throughout the grading process. While it can accommodate minor changes in table structures or attribute names, significant alterations may pose challenges to accurate query processing. Nevertheless, the system is designed to adapt to evolving database structures with minimal disruption to its functionality.

Furthermore, the system assumes the correctness of input queries provided by instructors. It relies on the accuracy of these queries rather than student submissions, streamlining the grading process and ensuring consistency in evaluation standards across assessments.

One notable limitation of our system is its exclusive support for Oracle DBMS. While it offers functionality tailored to Oracle environments, it lacks compatibility with other database platforms such as MySQL, PostgreSQL, or SQL Server. As a result, users relying on these alternative platforms may encounter compatibility issues or limitations in accessing advanced features specific to their chosen database systems.

Moreover, the system's automation capabilities are not exhaustive, and certain transformations may be constrained. For instance, while the system can handle basic transformations and optimizations, it may not fully automate more complex tasks. For instance, the system currently lacks the capability to transform left and right joins into other join types that yield identical results. Additionally, it does not support the aggregation of multiple functions into a single function, even when those functions produce the same result. Future iterations of the system are envisioned to address these constraints through enhanced automation and expanded functionality."

B. Analysis of standard functions provided by Oracle DBS

1) Handling errors in functions: In the context of Oracle DBS, errors in functions can often arise due to typographical errors or misspellings. To mitigate this issue, an advanced approach is employed, utilizing the Levenshtein distance algorithm [7] . This algorithm serves to identify the closest matching function to the input, enabling the system to automatically correct potential typos. Additionally, to streamline the management of functions, a structured approach is adopted, with functions organized and stored in three distinct JSON files. The first file contains a repository of the most frequently used functions, derived from recent entries. The second file encompasses functions that are not directly retrievable from the database. Lastly, the third file consists of functions that can be obtained via SELECT queries. Moreover, to ensure the integrity of function parameters, a thorough parameter check is conducted. This check verifies both the correctness of parameter types and their order within the function. In instances where parameters are found to be correctly specified but in the wrong order, the system reorders them to rectify the issue.

2) Transformation of functions with similar results: Within the Oracle DBS environment, exist functions that yield identical or similar results. For instance, functions like TO_CHAR and EXTRACT may produce similar outcomes for certain data transformations. To facilitate easier result comparison and analysis, a systematic approach to function transformation is implemented. Through code transformation techniques, the system is capable of converting between such similar functions seamlessly. This not only enhances the efficiency of result analysis but also streamlines the process of identifying optimal function usage within queries. Also, the user can use the test application to add two such functions that return the same result in a particular case, and then the functions are mapped to each other and during the run of the program they are converted into a single function if they occur.

C. Addressing errors in keywords

Keywords play a fundamental role in defining the syntax and structure of SQL queries. Common keywords such as SELECT, FROM, and WHERE are integral components of query construction. To ensure syntactic accuracy and logical coherence in query formation, a robust mapping system for keywords is established. This mapping system not only identifies potential keyword errors using the Levenshtein distance algorithm [7] but also considers the contextual relationships between keywords. By mapping out the permissible sequences of keywords, the system can effectively identify and rectify errors in keyword usage, thereby enhancing the overall robustness and accuracy of SQL queries.

D. Handling nested queries

1) Properly completing missing parentheses, quotation marks and apostrophes: Proper syntax is imperative in SQL query construction, particularly when dealing with nested queries. One common challenge is ensuring the completeness of parentheses, quotation marks, and apostrophes within queries. To address this challenge, the system employs a two-fold approach. Firstly, syntactic validation is performed using the jsqlparser library, which verifies the correctness of query syntax. Secondly, a logical algorithm is implemented to identify and complete missing parentheses and quotation marks. This algorithm leverages the inherent structure of SQL queries, including the minimum and maximum parameter counts associated with functions. By analyzing the query structure, the system can insert missing parentheses and quotation marks, ensuring syntactic integrity and query correctness.

2) Saving nested queries: Nested queries, which involve the embedding of one query within another, are prevalent in database operations. To manage nested queries efficiently, a structured approach to query storage is adopted. Within the system architecture, queries are organized and stored within a designated java class known as STATEMENT. This class incorporates parameters such as usedTables and functions, allowing for comprehensive query representation. Additionally, the STATEMENT class supports nested queries within nested queries, enabling the system to accommodate complex query structures seamlessly.

E. Typos in names of tables and columns

Typos in table and column names was one of the topics our colleague addressed. Typos in table and column names pose challenges in SQL queries, impacting query accuracy. This aspect of our system, developed by our colleague, utilizes the Levenshtein distance algorithm to identify typos. By calculating the similarity between words, potential typos can be detected. This algorithm forms the cornerstone of our approach, enabling the system to pinpoint discrepancies in names effectively. [7]

Incorporating support information enhances the accuracy of typo correction. For instance, when two columns within a table have same distances from the word with the typo, contextual clues can be pivotal. If a column appears in the GROUP BY clause, it signifies its importance and correctness. Leveraging this insight, our system ensures that the correct column is also included in the SELECT part of the query. By integrating contextual information from query structures, our system provides more reliable typo corrections, optimizing query execution and accuracy. [8]

F. Aliases

1) Aliases in the select section: This section introduces a concept developed by our colleague, Bc. Lukáš Jancik, focusing on aliases within the context of select section. The concept is elaborated upon briefly here. When there's an alias on a column name, we remove it to simplify comparing two statements at the end. This change applies recursively, even within nested queries, until we reach the primary query or certain clauses like ORDER BY or WHERE. If both queries within a nested structure use the same alias, the order by part follows the most deeply nested query, similar to how SQL Developer for Oracle handles it. This approach ensures that comparing statements remains straightforward and consistent, making it easier to analyse and understand any differences between them. [8]

2) Aliases for tables: If aliases are used over the tables, we simply append the alias before the column name (e.g., alias.column) to facilitate comparison later on. Additionally, as part of this process, potential errors arising from the absence of fully qualified attribute names can be identified. This error occurs only when an attribute exists in multiple tables and the column share the same name. It's a logical error, and the database system interprets it as an ORA-00918 error , indicating that the column is ambiguously defined. This situation arises specifically when a column exists in two or more joined tables with identical column names. [9]

G. Transformations of different types of joins

1) Join using and join on: Joins are fundamental operations in SQL for combining data from multiple tables. In Oracle DBS, there are different types of joins, each with its own syntax and usage conventions. One common transformation involves converting "join using" syntax to "join on" syntax. This transformation enhances query readability and flexibility by providing a more versatile approach to specifying join conditions. By standardizing join syntax across queries, the system simplifies query maintenance and optimization.

2) In to Exists: Another transformation involves converting "IN" statements to "EXISTS" statements where applicable. This transformation is particularly beneficial for optimizing query performance and clarity. Before executing this transformation, the system verifies that the "IN" statement is not part of a function and assesses its suitability for transformation. Upon confirmation, the system executes the transformation, thereby enhancing the efficiency and readability of SQL queries.

When considering the transformation from "IN" to "EXISTS," several factors come into play. The transformation is feasible in scenarios where the subquery operates independently and does not rely on data from the outer query (non-correlated subquery). Additionally, the subquery should not involve complex logic or aggregation functions that cannot be easily incorporated into the transformed query. Furthermore, performance implications, such as data volume and query optimization, should be carefully evaluated to ensure the transformation yields the desired performance improvements. It's essential to conduct thorough testing and analysis to validate the effectiveness of the transformation in each specific scenario.

H. Comparing statements and evaluating them

The final component of this application, 'Comparing Statements,' is a contribution from our colleague Bc. Lukáš Jancik. This section offers a comparative analysis of statements. Aligning the teacher's correct statement with the potentially correct statements provided by the students is a highly extensive task that demands a significant amount of analytical work. The primary challenge lies in the myriad ways students may attempt to express the correct response to a given task through the composition of specific SQL commands.

This section primarily focuses on constructing a set of equivalent commands. The larger this set (the more equivalents discovered), the greater the likelihood of confirming the correctness of the compared student command, even if it may differ significantly (yet still equate to the correct response to the task).

Some statements may vary in their expression but remain logically equivalent. For instance, a student's statement might utilize aliases over the tables, whereas the teacher's statement does not use any aliases. Another scenario arises when conditions are expressed in reverse order, such as name='PETER" and "PETER'=name. Similarly, in the case of joining two tables using 'join on', the tables in the join can be swapped, reflecting the same principle of logical equivalence through different implementations. [8]

I. Referencing system tables

System tables in Oracle DBS serve as repositories for vital metadata and configuration information essential for effective database management. To optimize access to this critical data, a structured approach to referencing system tables is meticulously adopted. These system tables, housing invaluable details crucial for database operations, are serialized and

meticulously stored in JSON files. This serialization process facilitates efficient data retrieval and storage, ensuring seamless access to system table information.

Updates to system table data are seamlessly integrated into the system, with automatic triggers activated whenever new data is introduced into the database. This ensures that the system maintains synchronized references to system tables, thus upholding the integrity and consistency of database operations.

Moreover, system tables play a pivotal role beyond mere metadata storage; they are also instrumental in retrieving information about various functions utilized within the Oracle DBS environment. By leveraging data from user tables, system tables provide comprehensive insights into the usage and performance of functions, enabling informed decision-making and optimization strategies.

J. API

The API serves as a pivotal component for analyzing the correctness of DML queries within the Oracle DBS environment, particularly in conjunction with the EXAM system at FRI, UNIZA. Connected to the EXAM system, where students write their tests, the API receives the results from these tests. It then undertakes transformations, error analysis, and performance evaluations on these results, returning scores, logs, and statistics to the users.

Built on the Spring Boot framework, the API offers a streamlined approach to query analysis, providing standardized endpoints and response formats for efficient and secure interactions. Leveraging advanced algorithms and query validation techniques, it assesses the syntactical and semantic accuracy of submitted queries, identifying errors or inconsistencies.

Furthermore, the API provides valuable insights into query execution and error analysis, enabling users to refine their query writing skills and optimize database operations effectively. By integrating with the EXAM system, it offers a seamless experience for students, instructors, and administrators, ensuring the integrity and accuracy of database interactions within the educational context at Faculty of Management Science and Informatics, UNIZA.

K. Run-time life cycle

The sequence diagram outlines the interaction flow between a Testing Application, an API, and a Database. The process initiates with the Testing Application triggering actions like adding functions, starting developer mode, and processing data. Subsequently, the API responds with actions like changing configuration files for grading and sending files.

The main flow begins with sending request from Testing application to API to process one or two SQL statements. If it is the initial run of the application, then the tables and functions are retrieved from database and saved into JSON files. Following the initial steps, the subsequent phase involves a process characterized by the execution of numerous functions and transformations within the API. Upon completion of this phase, the API furnishes the Testing application with the transformed SQL statement, accompanied by logging information and a score.

The Testing Application also interacts with the API by initiating developer mode and retrieving data such as tables and functions. API then proceeds to save this information. Following these requests, the Database processes the incoming data and responds accordingly.

Overall, the sequence diagram illustrates a cohesive flow of actions between the Testing Application, API, and Database, showcasing the interactions involved in the testing process.

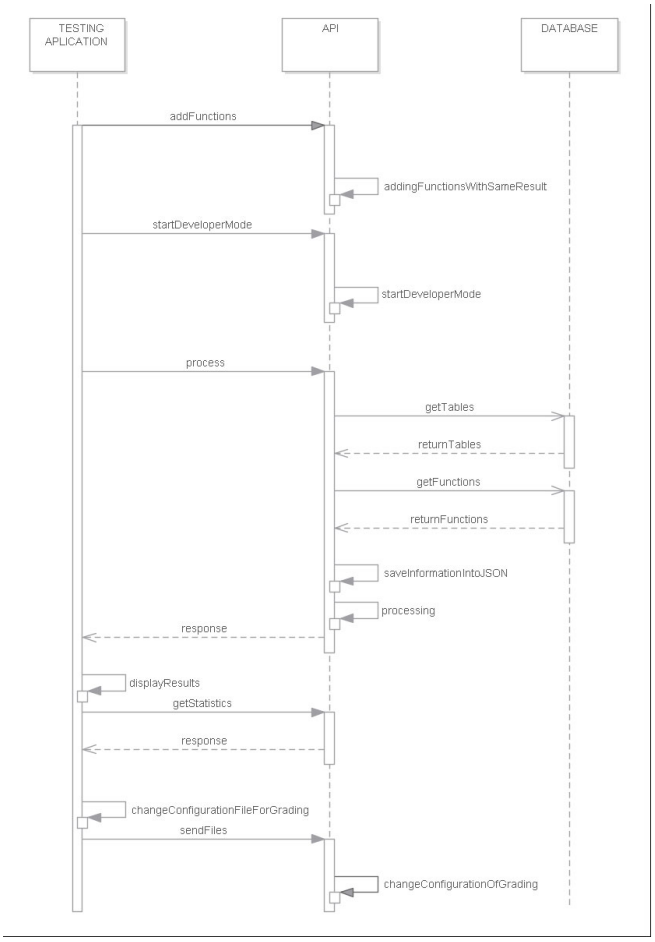

Fig. 1. Sequence Diagram for whole solution

L. Testing application

The testing application operates in conjunction with the API, providing a crucial platform for displaying transformed queries and facilitating comparison.

Through the testing application, users can submit SQL queries, including those generated by students, to be processed by the API. The application offers a user-friendly interface for inputting queries and initiating the transformation process.

A key feature of the testing application is its ability to display transformed queries generated by the API. Users can observe these transformed queries in real-time, gaining

insights into the impact of transformation techniques on query structure and syntax.

Furthermore, the testing application enables users to compare original and transformed queries side by side, allowing for thorough analysis of the transformation process. This comparison functionality provides users with valuable information to assess the effectiveness of transformation techniques and identify potential optimizations.

Additionally, the testing application offers comprehensive logging and error analysis capabilities. Users can review detailed logs of query transformations and any encountered errors, aiding in the refinement of transformation strategies and ensuring the reliability of the Oracle DBS environment.

In summary, the testing application serves as a valuable tool for displaying transformed queries and facilitating comparison, empowering users to evaluate the efficacy of transformation techniques within the Oracle DBS environment.

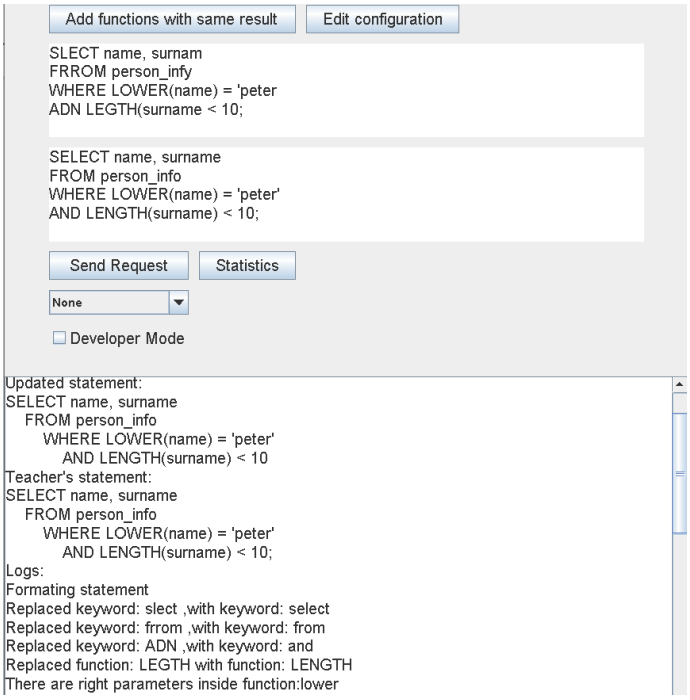

Fig. 2. Sample output in the test application

IV. TESTING AND ANALYSIS

A. Manual Testing

Manual testing is an essential and interactive process used to assess the functionality, usability, and performance of the API. Testers engage in exploratory testing, where they actively interact with the interface, navigating through functionalities, inputting diverse queries, and closely observing system responses. This hands-on approach uncovers unexpected behaviors, edge cases, and usability intricacies that may not be captured by scripted test cases.

An integral part of manual testing involves evaluating the interface's error handling mechanisms. Testers intentionally trigger errors to observe the system's response, assessing the effectiveness of error reporting, exception handling, and data integrity preservation.

While manual testing may not encompass exhaustive performance testing, testers monitor responsiveness and efficiency under varying conditions. They analyse response times, identify performance bottlenecks, and evaluate resource utilization to assess scalability and efficacy.

In practical terms, manual testing entails exploring different functionalities, sending diverse queries, and interacting with the interface in various ways to uncover errors. For example, in student queries, the interface may return partial transformations with logs, indicating errors that would typically be detected by SQL Developer or other Integrated Development Environments (IDEs).

B. Analysis of test results

Following manual testing, a thorough analysis of test results is conducted to identify areas for improvement and optimization. This analysis includes evaluating changes such as adjusting the loading order of JSON files or implementing optimizations like breaking for loops.

For instance, altering the sequence in which JSON files are loaded can significantly impact the interface's performance. By prioritizing the loading of frequently used files, such as those containing common functions, the interface can expedite query execution and enhance overall efficiency.

Similarly, implementing optimizations like breaking for loops can streamline processing and improve response times. By identifying and addressing such opportunities for enhancement, the interface can deliver a more seamless user experience and optimize performance.

Overall, the analysis of test results serves as a valuable feedback mechanism, driving continuous improvement initiatives to refine the functionality, usability, and performance of the API.

V. FUTURE WORK

In our commitment to ongoing enhancement, we are focusing on extending the integration of our solution to encompass popular Learning Management Systems (LMS). This expansion will facilitate seamless access for students and educators to view their scores and assessment outcomes directly within their preferred LMS interface. By incorporating this feature, we aim to optimize the educational process, fostering a smoother learning experience for all stakeholders.

Moreover, enhancing support for multiple database platforms is imperative for maximizing the versatility and applicability of our solution. Although currently optimized for Oracle Database Management System (DBMS), extending compatibility to widely used platforms such as MySQL, PostgreSQL, or SQL Server will significantly broaden its reach. This endeavor entails adapting the system to seamlessly accommodate the specific syntax, functions, and features of different database systems. By ensuring compatibility across

various dialects, we aim to promote consistency and efficiency in educational assessments across different database platforms.

Additionally, our focus on advancing automation capabilities remains unwavering as we continue to evolve. Specifically, in handling complex transformations and optimizations, our aim is to further streamline the grading process for educators. For instance, implementing additional transformations for joins, such as left and right joins, will enhance query performance and efficiency. Likewise, consolidating multiple functions into a single function when they produce identical results will simplify query structures and improve readability. These advancements will enable educators to allocate more time to teaching and guiding students, rather than manually grading tasks.

In summary, our future extensions aim to make our solution even more accessible, versatile, and efficient. By expanding integration with LMS platforms, enhancing support for multiple database platforms, and advancing automation capabilities, we are committed to providing educators and students with a comprehensive toolset that optimizes the educational assessment process.

VI. Conclusion

This section presents the key findings from our efforts to improve how we handle Data Manipulation Language (DML) commands, especially in educational settings like the Faculty of Management Science and Informatics, University of Zilina. Rather than enumerating all the discussed points, the emphasis will be placed on elucidating the principal insights and progress made in automating and improving educational assessments, especially when it comes to databases.

Dealing with DML commands can be tricky, especially when it comes to grading students' work. Traditional grading methods can be slow and prone to errors. In response, we've developed a tool to streamline the analysis and correction of DML commands within the EXAM system.

Our tool offers several features to make grading easier and more accurate. For example, it can analyse Oracle database functions, integrate smoothly with existing exam systems, and handle various types of SQL commands.

We also looked at existing tools and methods for grading DML commands. While they're helpful, they often fall short in providing a complete solution. Our tool aims to fill those gaps and provide a more comprehensive way to assess students' work.

Close collaboration was established with educators and stakeholders affiliated with the Faculty of Management and Informatics at UNIZA for the development of this solution. It's designed to make grading faster and more efficient, especially for tasks like evaluating SQL syntax and query optimization.

Future research perspectives could focus on addressing several key areas that remain unresolved and warrant attention in the immediate future. One crucial direction is the integration of this system into, ensuring seamless connectivity and utilization of its capabilities within academic settings. Additionally, there's a need for more comprehensive testing on real-world datasets to validate the efficacy and robustness of the system under diverse conditions. Although several short-term goals have been accomplished, as delineated in this manuscript, additional short-term objectives may encompass enhancing the efficiency, accuracy, and reliability of the grading process for students. This could involve refining aggregation functions, completing the program output for teachers, and addressing similar tasks to enhance the overall functionality and usability of the system. These endeavors will contribute to enhancing the functionality and practical applicability of the system, paving the way for more efficient and effective educational practices.

Comparing statements represents the most extensive area addressed in our work. Within this domain, there are abundant opportunities for analytical work and unexplored possibilities. This presents long-term potential and serves as fertile ground for ongoing research and development efforts.

Our solution is inherently adaptable to accommodate various SQL dialects, not only SQL Oracle. Whether catering to MySQL, PostgreSQL, or other database platforms, its architecture is primed for extension and integration. By embracing a broader spectrum of dialects, we can enhance its utility across diverse educational contexts and promote inclusivity within database education.

Overall, our solution represents a significant step forward in automating educational assessments, particularly in database-related courses. Through the utilization of contemporary technology and feedback garnered from educators, endeavors are being made to enhance the learning experience for both students and instructors.

Acknowledgment

This paper study was supported by the Erasmus+ project:

- Project number: 2022-1-SK01-KA220-HED-000089149, Project title: Including EVERyone in GREEN Data Analysis (EVERGREEN) funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the Slovak Academic Association for International Cooperation (SAAIC). Neither the European Union nor SAAIC can be held responsible for them.

References

[1] DbVisualizer official website, Formatting SQL, Web: https://confluence.dbvis.com/display/UG110/Formatting+SQL.

[2] JSQLParser official git, RDBMS agnostic SQL statement parser, Web: https://github.com/JSQLParser/JSqlParser.

[3] Charles Boisvert, Konstantinos Domdouzis, Joshua License, "A Comparative Analysis of Student SQL and Relational Database Knowledge Using Automated Grading Tools," IEEE, [Online]. Web: https://ieeexplore.ieee.org/document/8586684.

[4] Amol Bhangdiya, Bikash Chandra, Biplab Kar, Bharath Radhakrishnan, K. V. Maheshwara Reddy, Shetal Shah, and S. Sudarshan, "The XDa-TA system for automated grading of SQL

query assignments" IEEE, [Online]. Web: https://ieeexplore.ieee.org/document/7113403.
[5] Bikash Chandra, Ananyo Banerjee, Udbhas Hazra, Mathew Joseph, S. Sudarshan, " Automated Grading of SQL Queries" IEEE, [Online]. Web: https://ieeexplore.ieee.org/document/8731495.
Goran Đambić, Mario Fabijanić, Ana Lokas Ćošković, " Automatic, Configurable and Partial Assessment of Student SQL Queries with Joins and Groupings" IEEE, [Online]. Web: https://ieeexplore.ieee.org/document/9596680.
[6] B. Berger, M. S. Waterman, and Yun William Yu, "Levenshtein Distance, Sequence Comparison and Biological Database Search"., IEEE Trans Inf Theory. 2021 Jun; 67(6): 3287–3294.
[7] Bc. L. Jancik, "Analytical Tool for ORACLE SQL Statements", unpublished.
[8] M. Kvet, K. Matiaško, Š. Toth, *Practical sql for oracle cloud*. Žilina: EDIS-vydavateľstvo UNIZA, 2022.