JURNAL TEKNOLOGI DAN OPEN SOURCE
Vol. 9, No. 1, June 2026, pp. 79~87
e-ISSN: 2622-1659, accredited Four Grade by Kemenristekdikti, Decree No: 152/E/KPT/2023
DOI: https://doi.org/10.36378/jtos.v9i1.5474

Rule-Based SQL Grammar Validator

Syahrul Fajar Laqsono[1], Agung Prasetya[2], Mohamad Khoirul Ansor[3]
Program Studi Informatika, Fakultas Sains dan Teknologi, Universitas Bhineka PGRI Tulungagung

Article Info

Article history:
Received 04 28, 2026
Revised 05 19, 2026
Accepted 06 06, 2026

Keywords:
Texttosql
Grammarchecker
Sqlvalidation
Rulebased

ABSTRACT

This study addresses the growing need for reliable data access systems by focusing on the validation of SQL queries generated from natural language using a Text-to-SQL approach. The primary objective of this research is to evaluate the effectiveness of a rule-based SQL grammar validator in detecting syntactic errors and improving the overall quality of queries generated by Large Language Models (LLMs), particularly in the context of Indonesian language input. The research methodology follows a structured process, including literature review, dataset construction, system design, implementation, and performance evaluation. Two datasets were developed: one for validating the grammar checker using both valid and invalid SQL queries, and another for evaluating the Text-to-SQL system. The validator was implemented using a rule-based system with grammar defined in EBNF and executed using forward chaining inference. The results indicate that the system achieves high performance, with an accuracy of 0.909, precision of 0.857, recall of 1.000, and F1-score of 0.923. The validator successfully identifies common structural errors such as missing table references and incomplete JOIN clauses. However, some limitations remain in detecting more complex syntax patterns. Overall, the integration of the grammar checker significantly enhances the reliability of SQL query generation. In conclusion, the proposed system demonstrates strong effectiveness in syntax validation and contributes to improving the robustness of Text-to-SQL systems.

Corresponding Author:
Syahrul Fajar Laqsono
Program Studi Informatika
Universitas Bhineka PGRI Tulungagung
Jawa Timur, Indonesia
Email:fajarlaqsono08@gmail.com

1. Introduction

The rapid advancement of information technology has significantly increased the demand for efficient and accessible data management systems. Databases serve as fundamental components for storing and processing information across various domains, ranging from business to education [1]. Interactions between users and databases are generally conducted through query languages such as SQL, which follow specific syntax rules and structural requirements [2]. This situation often presents challenges for non-technical users, as it requires specialized knowledge to construct queries accurately and effectively. These limitations have encouraged the development of solutions that simplify communication between humans and database systems.

The Text-to-SQL approach has emerged as one of the most promising solutions to bridge this gap by automatically translating natural language into SQL queries [3]. This system enables users to express their data requirements using everyday language without needing to understand the technical details of SQL [4]. The translation process is performed through natural language processing techniques that allow the system to interpret the intent behind user questions. This capability significantly enhances data accessibility,

particularly for users without a technical background. Consequently, Text-to-SQL systems have become increasingly important in supporting data-driven decision-making across various sectors [5].

Advancements in artificial intelligence, particularly in natural language processing, have further improved the capabilities of Text-to-SQL systems [6]. The adoption of Large Language Models (LLMs) enables these systems to understand linguistic context more comprehensively and generate more complex and relevant SQL queries. These models are trained on large-scale datasets, allowing them to recognize language patterns and conceptual relationships within sentences. Such capabilities have established LLMs as a core component in the development of more accurate natural language-to-SQL translation systems [7] [8] [9]. This progress reflects a shift from traditional rule-based approaches toward more adaptive machine learning-based methods.

Numerous studies have been conducted to improve the performance of LLM-based Text-to-SQL systems through various approaches. Strategies such as domain-specific fine-tuning, prompt engineering techniques, and the incorporation of database schema information have been shown to enhance query quality [10] [11]. Other methods integrate execution-based validation mechanisms to ensure that generated queries can be executed correctly. Automated correction approaches have also been employed to address errors that occur during the query generation process [12] [13]. The findings from these studies demonstrate significant improvements in accuracy compared to previous methods.

The flexibility of LLMs has made Text-to-SQL systems increasingly adaptive in handling diverse user queries [14]. These models are capable of interpreting the intent behind complex sentences and mapping it to the available database structure. This process involves contextual understanding, entity mapping, and the selection of appropriate SQL functions according to user requirements [15]. As a result, the generated outputs are generally more relevant and closely aligned with user expectations than those produced by traditional approaches. This flexibility represents one of the primary advantages of applying LLMs to Text-to-SQL systems [16].

Despite these advancements, limitations remain regarding the validity of queries generated by LLMs. The system may occasionally produce SQL commands that do not conform to the required syntax or grammar rules. Errors can occur in the form of incorrect clause ordering, incomplete keyword usage, or inappropriate references to tables and columns. These issues indicate that strong language understanding does not always guarantee accurate query construction. Therefore, ensuring query validity remains a significant challenge in the development of reliable Text-to-SQL systems.

The impact of SQL syntax errors can affect overall system performance [17]. Invalid queries may result in execution failures, preventing the retrieval of the required data. Incorrect query outputs can also lead to misinterpretations during decision-making processes [18]. Another potential consequence is the rejection of queries by the database management system due to structural inconsistencies [19] [20]. These challenges highlight the importance of implementing additional mechanisms capable of verifying query validity before execution.

A rule-based approach can be utilized to address these syntax validation issues. Such a system operates by comparing query structures against predefined SQL grammar rules. Each query element is systematically examined to ensure compliance with the applicable standards. The primary advantage of this approach lies in its deterministic nature, which enables consistent and explainable validation results. Consequently, a rule-based system provides an effective complement to the limitations of LLM-based models.

An SQL grammar validator functions as an additional component within the Text-to-SQL pipeline, responsible for verifying the structural validity of generated queries. The validation process can be performed after query generation by the LLM as a post-processing stage. The validator identifies syntax errors and provides information regarding the non-compliant components of the query. This mechanism helps improve query quality before execution within the database system. The integration of a grammar validator therefore offers an additional layer of reliability to the overall system.

This study proposes the development of a rule-based SQL grammar validator integrated into an Indonesian-language Text-to-SQL system. The focus on the Indonesian language is motivated by the limited availability of related research addressing local needs. The dataset will be designed to represent various forms of natural language commands and simple database structures. The proposed validator will be evaluated to measure its effectiveness in reducing syntax errors and improving the quality of SQL queries generated by LLMs. The findings of this study are expected to contribute to the development of more reliable Text-to-SQL systems that are better aligned with the needs and contexts of local users.

2. Research Method

This research was conducted through a series of structured and interconnected stages to ensure the systematic achievement of the research objectives. The process began with a literature review to establish a comprehensive understanding of relevant concepts, including Text-to-SQL systems, SQL grammar, rule-

based systems, and model evaluation methods. The next stage involved dataset construction, consisting of pairs of Indonesian natural language texts and SQL queries, including both syntactically valid and invalid queries. Subsequently, the system architecture was designed by defining the SQL grammar rules using Extended Backus–Naur Form (EBNF) and specifying the validator workflow. The designed model was then implemented as a software system using the Python programming language, supported by rule-based libraries. The following stage involved system testing using the prepared dataset to evaluate the validator's ability to detect syntax errors. Performance evaluation was conducted using classification metrics such as accuracy, precision, recall, and F1-score to provide a comprehensive assessment of the model's effectiveness. The research process concluded with the preparation of a final report presenting the results, analysis, and discussion in a comprehensive manner.

The dataset used in this study was divided into two main parts to support a thorough evaluation process. The first dataset was designed to assess the performance of the SQL grammar checker in distinguishing between valid and invalid queries. This dataset was developed through several stages, including the extraction of initial query templates, the generation of structural query variations, the population of templates with table and column names, and a filtering process based on execution against a test database. Syntax errors were automatically introduced through an error injection technique to generate invalid queries containing various types of syntactic mistakes. The second dataset was used to evaluate the performance of the LLM-based Text-to-SQL system in the Indonesian language context. This dataset was constructed through database schema creation, annotation of question–SQL query pairs, and validation and paraphrasing processes to increase linguistic diversity.

The grammar validation system was designed using a rule-based approach by translating SQL grammar specifications expressed in EBNF into IF–THEN rules executed through a forward-chaining inference mechanism. The implementation was carried out using Python and the Experta library as the inference engine. An LLM was employed to generate SQL queries from natural language inputs, after which the generated queries were validated by the grammar checker. The evaluation focused not only on the validator's ability to identify syntax errors but also on its contribution to improving the quality of SQL queries generated by the LLM. This improvement was measured by comparing system performance before and after validator integration using additional metrics such as Execution Accuracy and Exact Match Accuracy.

3. Result and Discussion

This study was conducted to address the research problem concerning the effectiveness of a rule-based SQL grammar validator in detecting syntax errors within queries generated by Text-to-SQL systems. The entire research process followed a systematically designed methodology, beginning with dataset construction, continuing through system architecture design, and concluding with performance evaluation. The implementation focused on developing a deterministic validation mechanism, ensuring that every decision produced by the system could be explained based on the underlying rules. This approach not only determines whether a query is valid or invalid but also provides analytical insights into the causes of identified errors. The results obtained were analyzed comprehensively to assess the reliability of the proposed system and its contribution to improving the quality of SQL queries generated by Text-to-SQL models, particularly in the context of the Indonesian language.

The SQL Grammar Checker was developed using a rule-based approach that relies on a set of formal rules as the basis for decision-making. The system architecture was designed in a modular manner to support a structured and progressive validation process. The initial stage begins with tokenization, which utilizes regular expressions to decompose SQL queries into lexical units such as keywords, operators, identifiers, and literals. The resulting tokens serve not only as an initial representation of the query but also as the foundation for constructing facts used during the inference process. These facts are subsequently stored in working memory, enabling the system to efficiently and consistently perform pattern matching against predefined rules.

The inference engine was implemented using the Experta library, which supports forward chaining as the primary reasoning strategy. Each SQL grammar rule, originally specified in Extended Backus–Naur Form (EBNF), was translated into condition-based rules using the @Rule decorator. This approach allows the system to perform incremental evaluation by matching available facts against relevant rules. When a rule is satisfied, the system generates a conclusion regarding the validity of the query or identifies a specific syntax error. This mechanism provides a significant advantage in terms of transparency, as every decision can be traced back to the corresponding activated rule. The inference process proceeds iteratively until no additional rules can be executed, resulting in a final and consistent validation outcome.

The validation process implemented within the system was designed comprehensively to ensure that every generated SQL query complies with applicable syntax rules. Validation is not limited to checking the presence of essential query elements but also examines the relationships between query components as a

whole. Core clauses such as SELECT and FROM serve as the starting point of the validation process because they constitute the foundation of SQL query structure. The system verifies that both clauses are present and arranged in the correct order according to SQL standards. Through this approach, fundamental errors can be detected at an early stage before more detailed examinations of other query components are performed.

In addition to verifying mandatory clauses, the system validates the ordering of query clauses to ensure compliance with correct SQL syntax structures. Clauses such as SELECT, FROM, WHERE, GROUP BY, and HAVING must follow a predefined sequence, and any deviation from this order is immediately identified as an error. This validation is particularly important because clause-ordering mistakes are often overlooked by users, especially when constructing complex queries. By incorporating this mechanism, the system maintains structural consistency and prevents errors that could potentially lead to execution failures within database management systems.

The system is also equipped with the capability to detect a variety of common SQL writing errors. These include the absence of a table name following the FROM clause, the use of JOIN operations without a corresponding ON clause, and incorrect conditional expressions within the WHERE clause. Detection of these errors is performed through the application of predefined pattern-based rules. Consequently, the system not only identifies errors in a general sense but also provides more specific indications regarding the problematic portions of a query. This capability greatly assists both users and downstream systems in analyzing and correcting SQL queries.

Furthermore, the validation process encompasses additional clauses such as WHERE, GROUP BY, and HAVING, which are commonly used in more complex SQL queries. The system ensures that each clause is applied appropriately, both in terms of structural correctness and its relationship with other query components. For example, GROUP BY usage must be consistent with the selected columns, while the HAVING clause must be applied within the correct aggregation context. Through this broad validation coverage, the system provides assurance that every query undergoes a thorough examination before being classified as valid or invalid. Overall, this approach enhances the quality and reliability of SQL queries generated within the Text-to-SQL pipeline.

System performance was evaluated using a binary classification approach that categorizes queries into two primary classes: valid and invalid. The evaluation dataset consisted of a combination of verified valid queries and syntactically incorrect queries generated through an error injection process. This approach enabled the system to be tested across a variety of scenarios representative of real-world conditions. The evaluation process employed standard classification metrics, including accuracy, precision, recall, and F1-score, to provide a comprehensive assessment of system performance. The results indicate that the proposed system achieves a high level of effectiveness in detecting SQL syntax errors, particularly in its ability to consistently recognize valid queries.

The following section presents the system performance evaluation results based on the classification metrics employed in this study :

Table 1. Model Evaluation Results

Evaluation Metrics	Grade	Description
Accuracy	0.909	Accuracy of system predictions for all test samples
Precision	0.857	The reliability of the system in correctly labeling valid queries
Recall	1.000	The system's ability to detect all valid queries
F1-Score	0.923	Balance between precision and recall

An accuracy score of 0.909 indicates that the majority of the predictions generated by the system are consistent with the actual conditions in the test dataset. This achievement demonstrates that the proposed validation mechanism is capable of consistently distinguishing between valid and invalid SQL queries. The recall value of 1.000 indicates that all valid queries were successfully identified by the system, with no instances of misclassification into the invalid category. This result suggests that the system exhibits exceptionally high sensitivity toward legitimate queries, resulting in the absence of false negatives. In other words, every valid query within the evaluation dataset was correctly recognized as valid by the grammar checker.

The precision score of 0.857 indicates that a number of invalid queries were incorrectly classified as valid. This phenomenon resulted in the occurrence of false positives, suggesting that the implemented rule set has not yet fully captured all possible forms of SQL syntax errors. The difference between the recall and

precision values provides important insight into the behavior of the system. Specifically, the validator tends to adopt a more permissive rather than restrictive validation strategy. While this characteristic minimizes the risk of rejecting valid queries, it also increases the likelihood that certain invalid queries may pass the validation process undetected. Consequently, further refinement of the grammar rules may be necessary to improve precision while maintaining the system's high recall performance.

A more detailed analysis of the correctly classified queries was conducted to evaluate the system's capability in handling various SQL query structures. The experimental results demonstrate that the system is capable of accurately identifying queries with relatively high levels of complexity. This capability is reflected in its successful validation of aggregation-related clauses such as COUNT, GROUP BY, and HAVING, as well as nested conditions involving multiple logical expressions. These findings indicate that the rules defined within the knowledge base effectively cover many of the common patterns frequently encountered in SQL query construction.

Furthermore, the system was able to recognize dependencies between query clauses and verify whether they were arranged according to standard SQL syntax requirements. This capability enabled the validator to evaluate not only the presence of individual query components but also the overall structural consistency of the query. The broad coverage of the implemented rules represents a key factor contributing to the system's high classification performance across different testing scenarios. Overall, these results demonstrate that the proposed rule-based grammar validator provides a reliable mechanism for SQL syntax validation and serves as an effective complement to Text-to-SQL systems.

Table 2. Sample Queries with Accurate Predictions

SQL Query Example	Current Status	System Prediction
SELECT name, age FROM users WHERE age BETWEEN 18 AND 30;	VALID	VALID
SELECT department, COUNT*() FROM users GROUP BY department HAVING COUNT()* > 5;	VALID	VALID
SELECT age FROM WHERE age > 18;	INVALID	INVALID
SELECT name FROM users LEFT JOIN orders;	INVALID	INVALID

The system's ability to detect syntax errors is clearly demonstrated through several test cases representing common mistakes in SQL query construction. The validator successfully identified queries that lacked a table name following the FROM clause, recognizing them as violations of fundamental SQL structural requirements. This result indicates that the rule enforcing the presence of a table reference has been effectively implemented within the knowledge base. Furthermore, the system was able to detect the use of JOIN operations without an accompanying ON clause, which constitutes a critical SQL rule for establishing relationships between tables. The successful detection of these two categories of errors demonstrates that the validator possesses strong capabilities in identifying structural mistakes that frequently occur in SQL queries, particularly those generated automatically by Text-to-SQL models. These findings suggest that the rule-based approach can consistently and reliably handle fundamental syntax errors, thereby enhancing the overall robustness of the validation process.

An evaluation of misclassified cases was conducted to identify the limitations of the system in handling more complex or less common error patterns. The analysis revealed that several queries that were actually invalid were incorrectly classified as valid by the validator. This issue arises because the implemented rules do not yet encompass all possible variations of syntax errors that may occur in SQL queries. Certain errors require more sophisticated analysis, such as validating contextual relationships between tokens or employing lookahead mechanisms during the parsing process. As a result, some syntactic inconsistencies may remain undetected despite the existence of extensive validation rules.

These findings indicate that although the current system provides broad rule coverage and demonstrates strong performance in detecting common syntax violations, there is still room for improvement. Further development is required to expand the knowledge base with more detailed and specialized rules capable of addressing a wider range of SQL syntax patterns. Enhancing the rule set would enable the system to detect more complex errors, improve precision, and reduce the occurrence

of false-positive classifications. Such improvements are essential for increasing the reliability of the validator and ensuring its effectiveness when integrated into real-world Text-to-SQL pipelines. Overall, the results highlight both the strengths and current limitations of the proposed rule-based approach, providing valuable directions for future research and system enhancement.

Table 3. Analysis of Queries with Inaccurate Predictions

Failed Query	Initial Prediction	Reason for Violation
SELECT FROM users;	VALID	SELECT_EMPTY_LIST
SELECT name FROM users WHERE age BETWEEN 10;	VALID	BETWEEN_REQUIRES_AND

The first type of error occurs because the system does not yet include a rule that explicitly validates the presence of a column list after the SELECT clause. At the initial stage, the system only checks for the existence of primary keywords such as SELECT and FROM without verifying whether the elements between them are properly filled. As a result, structurally incomplete queries may still be considered valid by the system. This condition indicates that syntax validation cannot rely solely on the presence of clauses, but must also consider the completeness and content of each query component. Therefore, adding more specific rules that target internal clause elements is an important step to improve validation accuracy.

Another limitation is observed in queries that use the BETWEEN operator without a complete AND-bound value pair. The system is unable to detect this error because it lacks a deeper inspection mechanism for relationships between tokens within a single conditional expression. Syntactically, the BETWEEN operator requires two boundary values connected by AND, and the absence of either component should be immediately classified as an error. The system's inability to recognize this pattern indicates that rule-based validation still needs to be extended with more detailed logic related to expression structure. Developing context-aware rules would be a potential solution to address this limitation.

These two types of errors suggest that the current rule coverage is still limited to basic and commonly occurring patterns. More complex syntax variations require a more detailed and systematic rule definition. The system must be capable of understanding not only clause ordering but also the logical relationships between query elements. Adding new rules that cover specific conditions would enhance the system's ability to detect a wider range of errors. This development should also consider error patterns commonly produced by LLMs to ensure better relevance to practical use cases.

The discussion of the results shows that a rule-based approach has significant advantages in terms of output consistency. The system always produces the same decision for identical inputs because it does not involve probabilistic elements like machine learning models. This characteristic is crucial in syntax validation, where determinism and clarity are required. Such consistency simplifies the testing process and ensures that validation results remain reliable under different conditions. This makes the rule-based approach a stable solution for environments that require high accuracy.

A major advantage of the rule-based approach lies in its high level of transparency in the validation process. Every decision is not a black-box output but can be clearly traced back to predefined rules. When a query is classified as valid or invalid, the system can indicate which rules were satisfied or violated. This provides a deeper understanding of the decision-making process, offering not only a label but also a logical explanation behind it.

This transparency is highly beneficial in analyzing SQL query errors. When an error is detected, developers or users can easily identify which part of the query caused the violation. This allows debugging to be performed more quickly and efficiently without manually examining the entire query structure. Thus, the system functions not only as a validator but also as an analytical tool that provides insights into syntax errors.

In addition, this approach simplifies system maintenance and further development. Since each rule is explicit and modular, modifications can be made directly to specific rules without affecting the entire system. Developers can add new rules, adjust existing ones, or remove irrelevant rules with greater flexibility. This makes rule-based systems more adaptable to continuous development needs, especially when dealing with increasingly complex query variations.

Furthermore, transparency in rule-based systems provides added value in terms of auditability and accountability. In environments that require clear decision-making processes, such as data-driven systems or critical applications, the ability to trace logical reasoning is essential. Each validation result can be systematically explained and well documented, thereby increasing trust in the system.

Finally, this approach also demonstrates strong stability across different input variations. The system does not require retraining when exposed to new data, as long as the input remains within the defined rule scope. This makes it more efficient to implement, as it does not depend on large training datasets. Such stability is

crucial in production environments where consistent performance is required without significant model adjustments. The system can be deployed directly without additional adaptation processes.

The integration of the SQL Grammar Checker into the Text-to-SQL pipeline provides a significant contribution to improving the quality of queries generated by the system. Prior to this integration, queries produced by the model were often directly forwarded to the database system without undergoing adequate syntax validation. This practice could potentially lead to various issues, including execution failures and query results that do not align with user requirements. With the presence of the grammar checker, every generated query is subjected to a validation stage before further processing. Only queries that satisfy the defined syntax rules are allowed to proceed to execution. This approach ensures that the quality of the output becomes more controlled, reliable, and consistent.

The validation process embedded within the pipeline functions as an essential safeguard mechanism. The SQL Grammar Checker acts as a preliminary filter capable of detecting and blocking erroneous queries before they reach the execution stage. This mechanism not only prevents errors at the database level but also protects data integrity from potential logical inconsistencies caused by invalid queries. As a result, the system becomes more robust due to the presence of an additional control layer that operates automatically in every natural language to SQL translation process.

The impact of this integration is also reflected in the improved overall reliability of the system. With a reduced number of failed query executions, system performance becomes more stable and dependable across different usage conditions. Furthermore, the risk of incorrect data retrieval is minimized, leading to more accurate results that better reflect user intent. This aspect is particularly important in data-driven decision-making contexts, where even minor query errors can significantly affect analytical outcomes.

Moreover, the integration of the Grammar SQL Checker reinforces the idea that validation is an inseparable component of an effective Text-to-SQL pipeline. The system is not only expected to translate natural language into SQL queries but must also ensure that the generated queries are syntactically correct and executable. The presence of a validator within the pipeline creates a balance between the generative capabilities of the model and rule-based quality control. Therefore, this integration represents an important step toward building a Text-to-SQL system that is more reliable, secure, and suitable for real-world deployment.

The validator also plays a crucial role in filtering errors that are not consistently detected by Large Language Models (LLMs). While LLMs are effective in understanding natural language, they may still produce SQL queries that violate syntactic rules. The grammar checker serves as a control mechanism that ensures model outputs comply with predefined standards. The combination of generative LLM capabilities and rule-based validation results in a more robust and dependable system. This synergy is a key factor in enhancing the overall quality of the generated queries.

The study also demonstrates that the use of the Indonesian language in a Text-to-SQL system can be effectively implemented. The grammar validator successfully supports the translation process from natural language to SQL by ensuring that query structures remain compliant with established rules. This finding indicates that the proposed approach is not language-dependent and can be adapted to local needs. The success of this implementation opens opportunities for developing similar systems in other languages using the same methodology, thereby adding significant value to language-based technology development.

Overall, the developed system shows strong performance in detecting SQL syntax errors. High evaluation scores and the ability to handle various query variations indicate the effectiveness of the proposed approach. Nevertheless, further improvements are still required to expand rule coverage so that more complex error patterns can be addressed. Enhancing rule specificity and strengthening validation mechanisms are expected to improve system accuracy. These future developments will make the system more adaptive and suitable for a broader range of real-world applications.

4. Conclusion

Based on the results of this study, it can be concluded that the rule-based Grammar SQL Checker system is effective in detecting syntax errors in SQL queries generated by Text-to-SQL systems. This is evidenced by high evaluation scores, particularly in accuracy, recall, and F1-score metrics, which indicate that the system is capable of reliably distinguishing between valid and invalid queries. The rule-based approach offers advantages in terms of consistency, transparency, and ease of error traceability, allowing every decision produced by the system to be logically explained based on the defined rules. In addition, the system is also proven to handle various query variations, including those with relatively complex structures.

However, this study also reveals certain limitations in the proposed system, particularly in detecting more complex or uncommon SQL syntax error patterns. Some invalid queries were still classified as valid due to incomplete rule coverage within the system. Therefore, further development is required, especially in expanding and refining the grammar rules as well as incorporating more context-aware validation

mechanisms. Overall, the integration of the Grammar SQL Checker into the Text-to-SQL pipeline provides a positive contribution to improving the quality and reliability of SQL query generation, while also opening opportunities for further development of similar systems for other natural languages, including Indonesian.

References

[1] M. A. M. B. Baso Dzulkifli Muhajir, "SISTEM MANAJEMEN BASIS DATA TERHADAP PERUBAHAN KARAKTER PEMBELAJARAN DI SEKOLAH MTS AN-NUR RANTEBARU," *J. Ilm. Pendidik. Dasar*, vol. 09, no. 03, 2024.

[2] M. Aslyza, "MANAJEMEN DATA BERBASIS DATABASE : SOLUSI UNTUK PENYIMPANAN DAN AKSES DATA YANG LEBIH EFISIEN," vol. 2, no. 3, pp. 909–917, 2025.

[3] J. T. Santoso, *SQL Structured Query Language*. 2021.

[4] Y. Salim and M. Hasnawi, "Konversi Bahasa Indonesia ke Perintah Data Manipulation Language pada Structured Query Language menggunakan Natural Language Processing," vol. 3, no. 3, pp. 181–187, 2022.

[5] R. S. U. Fitria Nur Hasanah, *BASIS DATA*. 2020.

[6] M. Amien, "Sejarah dan Perkembangan Teknik Natural Language Processing (NLP) Bahasa Indonesia : Tinjauan tentang sejarah , perkembangan teknologi , dan aplikasi NLP dalam bahasa Indonesia," pp. 99–105, 2023.

[7] A. P. Oka Alvansyah, Nezza Anggraini Yolandari, M. Fikri Zulfi, Afifah Naila Nasution, "PEMBUATAN WEBSITE PEMINJAMAN BUKU PERPUSTAKAAN DIGITAL DENGAN INTEGRASI FITUR KECERDASAN BUATAN UNTUK MENINGKATKAN LITERASI," vol. 3, no. 2, 2025.

[8] D. Z. Oktavia, D. A. Hidayat, D. Natalia, and S. K. Prabantara, "Machine Learning Performance Comparison for Web Application Security Threat Detection : A Systematic Review," vol. 5, no. 1, pp. 326–339, 2026.

[9] M. I. P. N. Nayla Arnona Br Damanik, "IMPLEMENTASI MANAJEMEN BASIS DATA DALAM LINGKUP KECERDASAN BUATAN," *KAMPUS Akad. PUBLISING*, vol. 2, no. 2, pp. 637–648, 2025.

[10] I. Frincu, "In Search of the Perfect Prompt A User Evaluation of Soft and Hard Prompt Tech- niques for Conversational Abstract Generation," 2023.

[11] R. R. Golani, "LLM Fine-Tuning vs Prompt Engineering for Consumer Products," vol. 16, no. 2, pp. 1–21.

[12] X. Zhang *et al.*, "Comparison of Prompt Engineering and Fine-Tuning Strategies in Large Language Models in the Classification of Clinical Notes 3 Department of Computer Science and Engineering , College of Engineering , Michigan State," 2024.

[13] I. W. Syahputri, E. K. Budiardjo, and P. O. H. Putra, "Unlocking the Potential of the Prompt Engineering Paradigm in Software Engineering : A Systematic Literature Review," 2025.

[14] C. D. Bui, H. H. Nguyen, and T. Q. Ngo, "A systematic survey of LLM-based text-to-SQL : methodologies , security vulnerabilities , and future challenges," pp. 1–83, 2026, doi: 10.7717/peerj-cs.3773.

[15] H. Bian *et al.*, "DKASQL : Dynamic Knowledge Adaptation for Domain-Specific," pp. 1–21, 2025.

[16] Z. Shao, S. Cai, R. Lin, and Z. Ming, "Enhancing Text-to-SQL with Question Classification and Multi-Agent Collaboration," pp. 4340–4349, 2025.

[17] S. Patulus *et al.*, "IMPLEMENTASI TEKNIK QUERY OPTIMIZATION UNTUK MENINGKATKAN," vol. 9, no. 2, pp. 2437–2442, 2025.

[18] Rachmawati, "ANALISIS KESALAHAN MENERAPKAN BAHASA SQL (STRUCTURE QUERY LANGUAGE) MATA KULIAH BASIS DATA Rachmawati IKIP Budi Utomo Malang Basis data diibaratkan sebagai arsip penyimpanan sebuah sistem Menurut Jogiyanto (2005) basis data adalah kumpulan dari data y," *J. Prism.*, vol. 1, no. 2, pp. 27–34, 2025.

[19] G. Rafianto, A. Voutama, U. S. Karawang, T. Timur, and J. Barat, "IMPLEMENTASI BASIS DATA TERSTRUKTUR DENGAN PENCEGAHAN SQL INJECTION PADA SISTEM," vol. 13, no. 2, 2025.

[20] Z. Apriza, "Tantangan dan Solusi Pengelolaan Basis Data : Dari Keamanan Hingga Optimalisasi Query Keamanan menjadi salah satu prioritas utama dalam sistem basis data modern , terutama," vol. 3, no. 2, pp. 448–454, 2025.

[21]. Nugraha, R., Abdul Rezha Efrat Najaf, & Reisa Permatasari. (2025). BCA Stock Price Prediction Using Time Series Method With GRU (Gated Recurrent Unit). JURNAL TEKNOLOGI DAN OPEN SOURCE, 8(2), 432 - 440. https://doi.org/10.36378/jtos.v8i2.4500

[22]. Mindara, G. P., Aisya Tyanafisya, Siti Farah Fakhirah, Azhar Nadhif Annaufal, Ibnu Aqil Mahendar, & Aditya Wicaksono. (2025). Design and Development of an E-Commerce Website Using the Waterfall Method with the Laravel Framework. JURNAL TEKNOLOGI DAN OPEN SOURCE, 8(2), 441 - 452. https://doi.org/10.36378/jtos.v8i2.4570

[23]. Siti Rodiyah, Sinulingga, S. M., Putra, F. R., Aryasatya, M. F., Kusdaryanto, A., Sulistiyono, R., & Irawan, B. (2025). Implementation of an Information System for Classroom Reservation at Esa Unggul University. JURNAL TEKNOLOGI DAN OPEN SOURCE, 8(2), 453 - 466. https://doi.org/10.36378/jtos.v8i2.4633

[24]. Joisce, Hanum Putri Kamelianti, & Rizal Aprianto. (2025). Concept And Potential For The Implementation Of Smart Parking System In Indonesia: A Literature Review . JURNAL TEKNOLOGI DAN OPEN SOURCE, 8(2), 467 - 481. https://doi.org/10.36378/jtos.v8i2.4647

[25]. Fariz, Eka Dyar Wahyuni, & Tri Luhur Indayanti Sugata. (2025). Implementation of the FP-Growth Algorithm for Bundling Strategy and Store Layout Redesign at Toko Kasih Ibu. JURNAL TEKNOLOGI DAN OPEN SOURCE, 8(2), 482 - 494. https://doi.org/10.36378/jtos.v8i2.4673

[26]. Windy Fadhilah Susanti, Ana Wati, S. F., & Indayanti Sugata, T. L. (2025). UI/UX Design of the “PrintOn!” Printing Services Marketplace for UMKM Photocopying and Printing Businesses Using the User Centered Design Method. JURNAL TEKNOLOGI DAN OPEN SOURCE, 8(2), 495 - 505. https://doi.org/10.36378/jtos.v8i2.4688

[27]. Kartika, A. D. P., Anindo Saka Fitri, & Nambi Sembilu. (2025). Developing a Web-Based Printing Transaction System Using The Prototype Method: A Case Study at Amanah Advertising. JURNAL TEKNOLOGI DAN OPEN SOURCE, 8(2), 514 - 528. https://doi.org/10.36378/jtos.v8i2.4692

[28]. Irawan, Y., Refni Wahyuni, & Herianto. (2025). An Integrated Machine Learning and Deep Learning Approach for Multiclass Flood Risk Classification with Feature Selection and Imbalanced Data Handling. JURNAL TEKNOLOGI DAN OPEN SOURCE, 8(2), 650 - 661. https://doi.org/10.36378/jtos.v8i2.4639

[29]. Revina Pravita Sari, Karnadi, & Jimmie. (2025). Applying the Waterfall method to build applications E-Commerce at Palembang City Computer Embroidery Partners. JURNAL TEKNOLOGI DAN OPEN SOURCE, 8(2), 1243 - 1256. https://doi.org/10.36378/jtos.v8i2.5199