NLP-DRIVEN TEXT-TO-SQL QUERY EXECUTION SYSTEM FOR NATURAL LANGUAGE DATABASE INTERACTION

Pinisetty Hanumath Satya Kiran[1], Mrs. B. Alekhya[2]
1UG Student, Department of Computer Science and Engineering
2Assistant Professor, Department of Computer Science and Engineering
Andhra Loyola Institute of Engineering and Technology, Vijayawada, Andhra Pradesh, India
Email: kiranpinisetty@gmail.com

Abstract: The widespread adoption of relational databases in data-driven organizations has created a significant barrier for non-technical users who lack proficiency in Structured Query Language (SQL). This paper presents an NLP-driven Text-to-SQL system that enables users to interact with MySQL databases using plain English queries, eliminating the need for manual SQL authoring. The proposed system integrates a React-based web interface with an ASP.NET Core backend and leverages an AI-assisted query generation engine powered by Groq AI. Unlike conventional approaches, the system operates on schema-aware query generation — only the database schema is shared with the AI engine, not the underlying data — ensuring both privacy and context-accurate SQL output. Users can authenticate securely, connect to their databases, explore schemas, and retrieve results through natural language inputs, which are automatically converted into validated, read-only SQL statements before execution. The system was evaluated across multiple custom database domains including banking, school management, and e-commerce, achieving a query translation accuracy of 80%–85%, with performance varying based on query complexity and prompt clarity. Experimental results demonstrate that the proposed system significantly simplifies database access for non-technical users while maintaining query safety and operational efficiency. This work highlights the potential of combining large language models with schema-aware generation to democratize relational database interaction.

Keywords: Text-to-SQL, Natural Language Processing, Natural Language Interface to Databases (NLIDB), Schema-Aware Query Generation, Groq AI, Large Language Models, Relational Database, SQL Automation

1. INTRODUCTION

The rapid growth of digital systems in recent years has led to an unprecedented increase in the volume of structured data stored and managed through relational databases. Organizations across domains such as banking, healthcare, education, and e-commerce rely heavily on these databases to store, retrieve, and analyze critical information. Accessing this information, however, requires users to formulate queries using Structured Query Language (SQL) — a powerful but technically

demanding language. SQL requires precise knowledge of query syntax, table structures, column names, and inter-table relationships. Even a minor error in syntax or logic can result in incorrect or failed query execution.

For non-technical users such as business analysts, managers, students, and domain experts, constructing SQL queries is a significant challenge. Remembering complex query structures involving multiple JOINs, nested subqueries, GROUP BY clauses, and aggregate functions is often impractical without formal training. As a result, these users remain dependent on database administrators or technical staff to retrieve even routine information, creating inefficiency and bottlenecks in data-driven workflows. According to a Stack Overflow developer survey, while SQL is used by over 51% of professional developers, only around 35% receive systematic training in it — highlighting a substantial knowledge gap even among professionals.

Several approaches have been explored to bridge this gap. Graphical query builders allow users to construct queries by selecting tables and columns through a visual interface, reducing the need to write SQL syntax manually. However, these tools still require users to understand the underlying database schema and the relationships between tables. Natural Language Interfaces to Databases (NLIDB) emerged as a more intuitive alternative, enabling users to pose questions in plain English. Early NLIDB systems relied on rule-based techniques and predefined templates that required manual engineering of grammar rules and keyword mappings. These systems struggled to handle ambiguous queries, complex sentence structures, and databases with dynamic or large schemas. Later, deep learning-based approaches using Seq2Seq models and LSTM networks improved accuracy, but these models require extensive domain-specific training data and often fail to generalize across different database schemas. Furthermore, most existing systems lack integrated features such as secure user authentication, real-time schema exploration, query validation, and safe execution controls — making them unsuitable for practical deployment.

To address these limitations, this paper proposes an NLP-driven Text-to-SQL system that leverages the power of large language models (LLMs) through Groq AI for intelligent, schema-aware SQL query generation. The proposed system provides a complete web-based solution built using React and ASP.NET Core, enabling users to connect to their MySQL databases, explore schemas, and retrieve results using plain English queries — without writing a single line of SQL. A key design decision is that only the database schema, not the actual data, is shared with the AI engine, ensuring data privacy while enabling contextually accurate query generation. All generated queries are normalized and restricted to read-only operations before execution, ensuring system safety.

2. LITERATURE SURVEY

Research in natural language interfaces to databases has evolved significantly over the past decade, progressing from rule-based systems to deep learning models and, more recently, to large language model (LLM)-based approaches. This section reviews key research contributions that form the foundation of Text-to-SQL systems and highlights the limitations that the proposed system aims to address.

Zhong et al. [1] introduced Seq2SQL, one of the earliest deep learning-based Text-to-SQL systems, along with the WikiSQL benchmark dataset comprising over 80,000 annotated question-SQL pairs. Seq2SQL applied reinforcement learning with policy-based optimization to generate SQL queries from natural language inputs and improved execution accuracy from 35.9% to 59.4% over earlier attention-based models. However, WikiSQL and the Seq2SQL model were restricted to single-table queries with simple SELECT-WHERE structures, limiting their applicability to real-world multi-table relational databases. Furthermore, the system operates purely as a research model without any integrated web interface, user authentication, or query execution pipeline.

Yu et al. [2] introduced the Spider dataset, a large-scale, cross-domain benchmark containing 10,181 natural language questions and 5,693 unique SQL queries across 200 databases with multiple tables covering 138 domains. Spider was designed to test the ability of models to generalize to new database schemas and complex SQL queries including JOINs, nested subqueries, and aggregation functions. The best performing model on Spider at the time of its release achieved only 12.4% exact matching accuracy, underscoring the difficulty of cross-domain, multi-table SQL generation. While Spider pushed the research frontier significantly, the models developed on it remain academic benchmarks without practical deployment features such as live database connectivity, secure user sessions, or result visualization.

Yu et al. [3] proposed SyntaxSQLNet, the first model specifically designed for the Spider benchmark, using syntax tree networks with nine independent sequence-to-set decoders to handle nested and complex SQL generation. SyntaxSQLNet outperformed prior approaches by 9.5% in exact matching accuracy on Spider. However, its overall accuracy remained below 30% on the full Spider dataset due to the inherent complexity of cross-domain schema generalization. The model requires large amounts of domain-specific annotated training data and cannot be directly applied to a user’s custom database without retraining.

Guo et al. [4] presented IRNet, which introduced an intermediate representation called SemQL to bridge the gap between natural language and SQL. IRNet addressed the schema linking problem — accurately mapping natural language tokens to the correct table columns — and achieved around a 20% improvement over SyntaxSQLNet on the Spider benchmark. Despite this

improvement, IRNet still relies on fixed training schemas and cannot dynamically adapt to new or user-defined databases without extensive fine-tuning.

Devlin et al. [5] introduced BERT (Bidirectional Encoder Representations from Transformers), a transformer-based pretrained language model that significantly advanced natural language understanding tasks. Subsequent work such as SQLova and X-SQL applied BERT as an encoder for Text-to-SQL generation, achieving state-of-the-art results on the WikiSQL benchmark. Similarly, Raffel et al. [6] proposed T5, a unified text-to-text transformer model that was fine-tuned for SQL generation and demonstrated strong performance on structured prediction tasks. While BERT and T5-based models improved translation accuracy, they require substantial computational resources for fine-tuning and still lack end-to-end practical deployment with live database connectivity, query validation, and result display.

Nihalani et al. [7] conducted an early review of natural language interfaces for databases, categorizing existing approaches into keyword-based, template-based, and semantic parsing methods. The review highlighted that flexible, schema-agnostic systems that can generalize across databases without manual rule creation were a critical unmet need — a gap that the proposed system directly targets through LLM-based schema-aware generation.

Zhu et al. [8] presented a comprehensive survey of LLM-enhanced Text-to-SQL generation, reviewing approaches that leverage large language models for zero-shot and few-shot SQL generation. The survey identified that while LLMs demonstrate remarkable generalization ability across schemas without retraining, a critical gap remains — the absence of end-to-end, production-ready systems that integrate LLM-based generation with practical database pipelines including query validation, secure authentication, and live execution. The proposed system addresses this gap by utilizing Groq AI — which internally runs state-of-the-art open-source LLMs such as LLaMA — as the query generation engine, combined with schema-aware prompting, read-only query validation, and a complete web-based deployment pipeline, delivering precisely the kind of practical system the survey identified as missing.

From the reviewed literature, the following key observations can be made:

- Early neural models such as Seq2SQL and SyntaxSQLNet are limited to fixed schemas and do not generalize to user-defined databases without retraining.
- BERT and T5-based systems improve accuracy but require large computational resources and domain-specific labeled data.
- Existing research systems operate as standalone models without integrated web interfaces, user authentication, or live database execution.

ISSN: 3049-2602

- Most systems expose database content to the AI model, raising data privacy concerns in real-world deployments.
- There is a clear research gap for a practical, schema-aware, LLM-based Text-to-SQL system that offers end-to-end deployment with query validation and security controls.

The proposed system addresses all of the above limitations by implementing a complete, deployable NLP-driven Text-to-SQL application that combines schema-aware LLM query generation, secure user authentication, real-time schema exploration, read-only query validation, and result display — features absent in prior academic systems.

3. PROPOSED SYSTEM

The proposed system, named DataExtractor, is a full-stack web-based NLP-driven Text-to-SQL application that enables non-technical users to interact with their MySQL databases using plain English queries. The system is designed with a three-layer architecture — a Client Layer, a Server Layer, and a Data Layer — each serving a distinct functional role in the end-to-end query pipeline.

Fig. 1 illustrates the system architecture. The Client Layer comprises the React-based Web UI that handles all user interactions including authentication, query input, schema exploration, and result display. The Server Layer is built on ASP.NET Core and contains the REST API, Authentication Service, Query Engine, and Schema Loader. The Data Layer consists of the Groq AI engine responsible for SQL generation and the MySQL database that stores user credentials and connected database schemas.

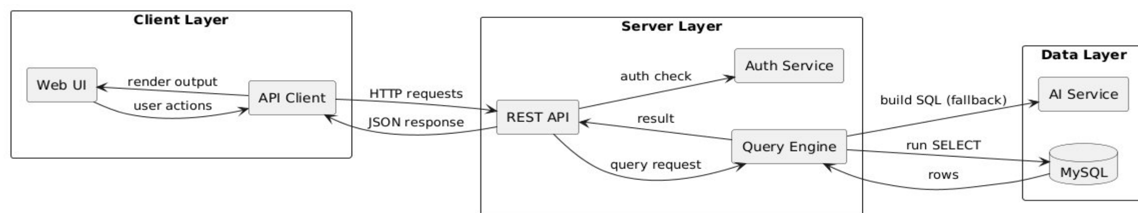

Fig. 1: System Architecture of DataExtractor

3.1 System Components

Authentication Module: The system provides a secure login and registration interface as shown in Fig. 2. Users must register and authenticate before accessing any database functionality. The branded login screen of DataExtractor accepts email/username and password credentials, with session management handled by the ASP.NET Core backend ensuring isolation of each user's data and connections.

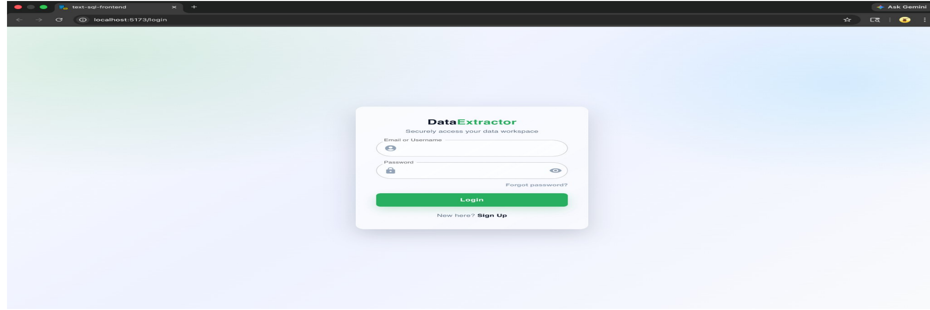

Fig. 2: User Authentication Interface

Database Configuration and Connection Module: After login, users configure their MySQL database connection through the Configuration panel as shown in Fig. 3, providing the host address, port (default 3306), connection name, and database credentials. The system supports saving previous connections for quick reloading. A single user can register and manage multiple MySQL server connections and switch between them at runtime using the Switch Database feature as shown in Fig. 4, which lists all available schemas — such as BankDb, ecommerce, and school_management — enabling seamless switching without requiring re-authentication.

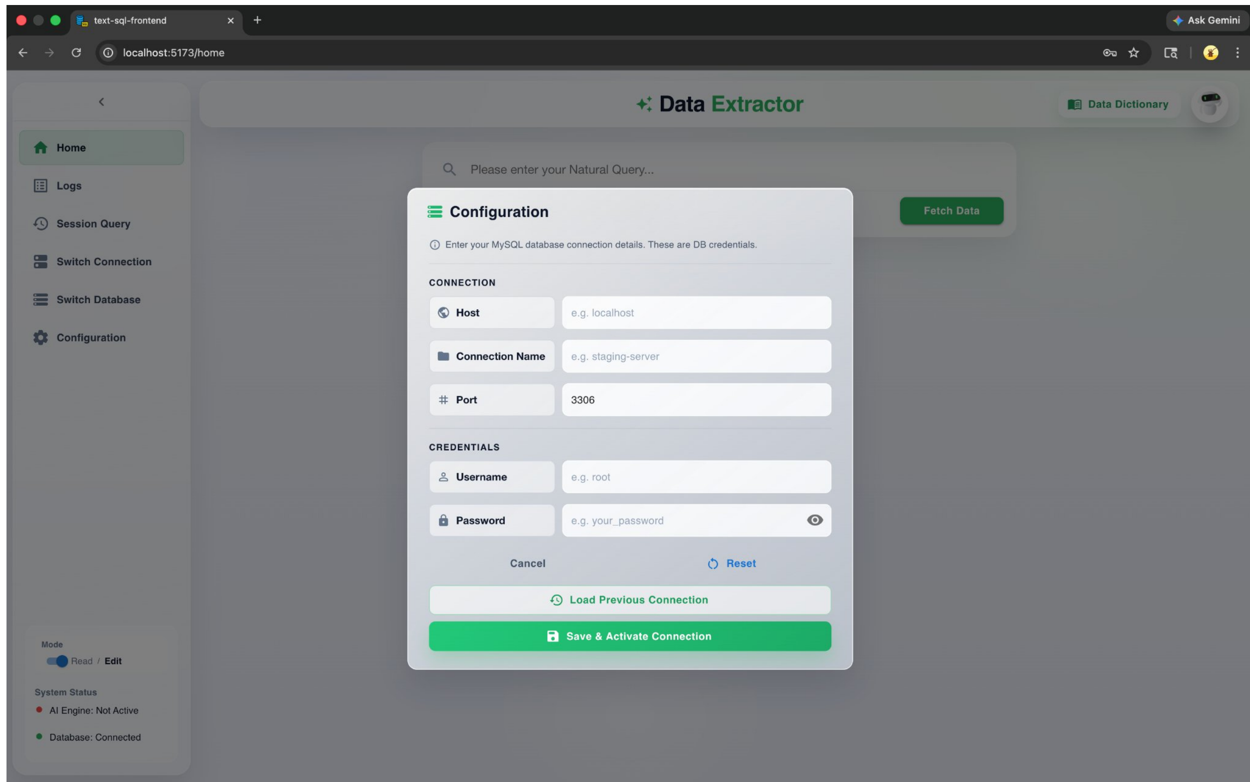

Fig. 3: Database Configuration Panel

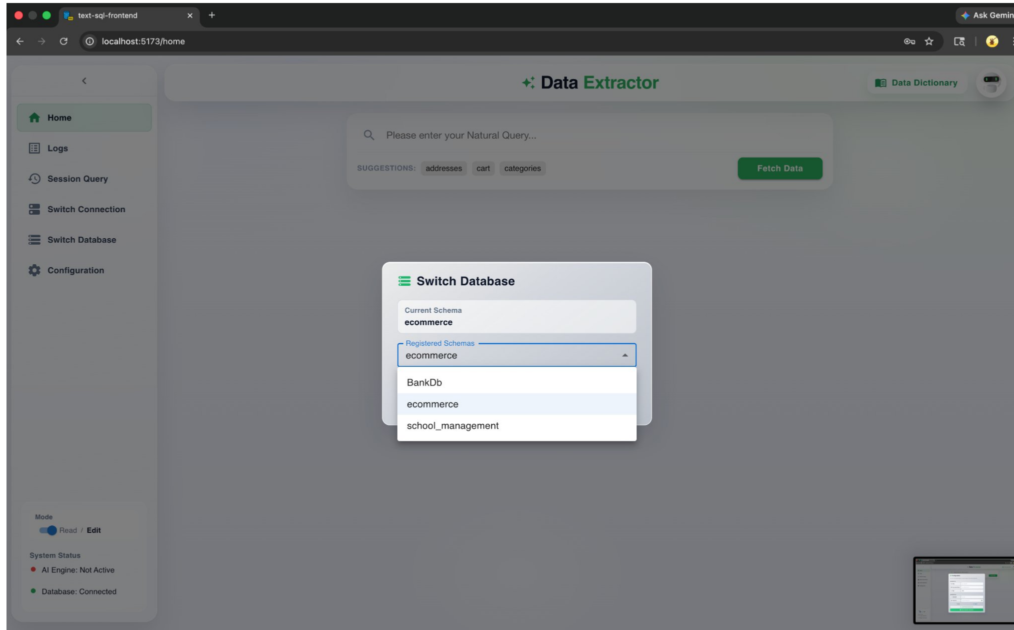

Fig. 4: Switch Database Module

Schema Explorer (Data Dictionary): A distinctive feature of the proposed system is the interactive Data Dictionary that provides users with a visual representation of the connected database schema as shown in Fig. 5. All tables are displayed as circular nodes labeled with their names and column counts. Users can click any table node to explore its columns and understand the database structure before formulating natural language queries. This empowers non-technical users to know what data is available and frame contextually accurate queries.

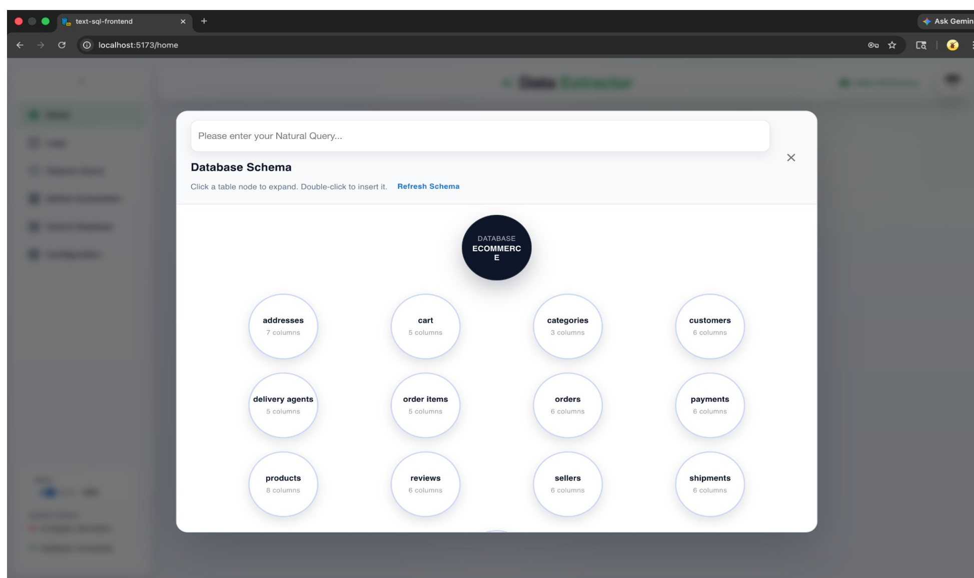

Fig. 5: Interactive Schema Explorer (Data Dictionary)

Journal of Computer and Electrical Sciences (JCES)- Volume-1 Issue-4, April 2026
ISSN: 3049-2602

Natural Language Query and Result Display: The home interface provides a central search bar where users enter queries in plain English, with table name suggestions and recent query history shown below for assistance. Upon submitting a query, the backend extracts the current database schema metadata and constructs a structured prompt for the Groq AI engine. Only the schema — table names, column names, and data types — is shared with the AI engine, never the actual data records, ensuring privacy. The AI engine generates a syntactically correct SQL query, which is then validated to permit only read-only SELECT operations before execution. Results are displayed in a structured tabular format showing row count, query latency, and column values as shown in Fig. 6.

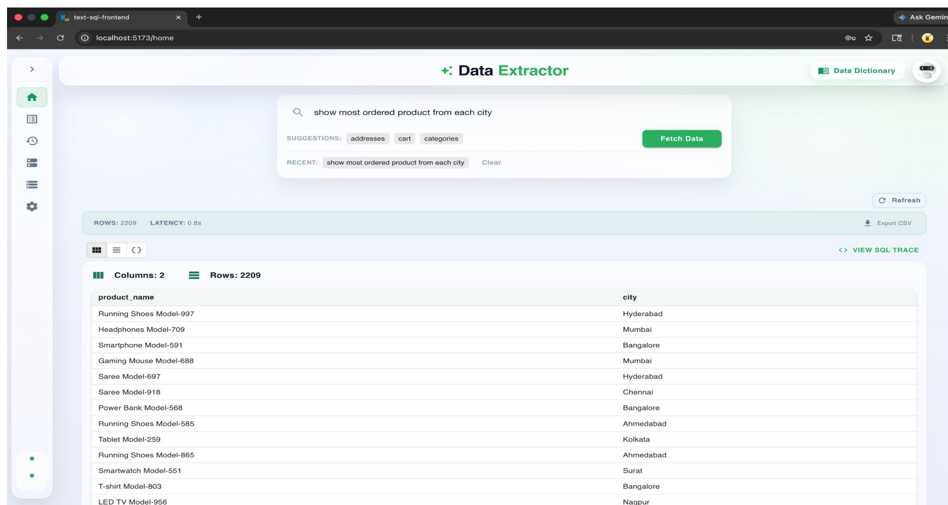

Fig. 6: Query Results — Natural Language Input with Tabular Output

SQL Trace (Query Transparency): To build user trust and support verification by technically inclined users, the system provides a View SQL Trace option that displays the exact SQL query generated by the AI engine as shown in Fig. 7. For example, a natural language query such as "show most ordered product from each city" is converted into a multi-table JOIN query with GROUP BY and ORDER BY clauses. This transparency feature distinguishes the system from black-box approaches and allows users to validate query correctness. Results can additionally be exported as a CSV file for further analysis.

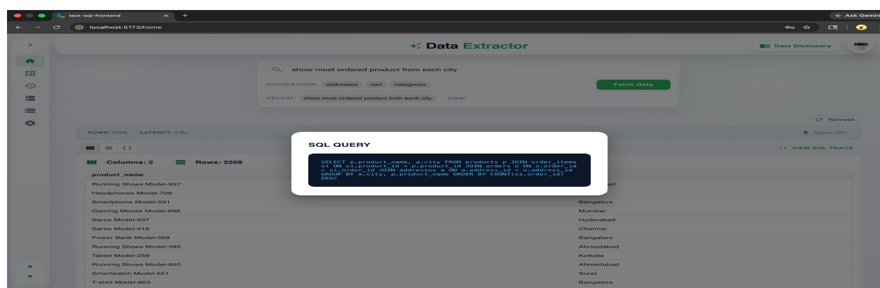

Fig. 7: SQL Trace — AI-Generated SQL Query Display

The system also includes a Logs module that maintains a history of all queries executed in the current session, and a Read/Edit mode toggle that controls whether the system operates in safe read-only mode or an extended mode for authorized users.

4. METHODOLOGY

The methodology of the proposed system defines the end-to-end processing pipeline that transforms a user's plain English input into an executed SQL query and a structured result. The pipeline is organized into seven sequential steps as described below.

Fig. 8 illustrates the complete query processing pipeline.

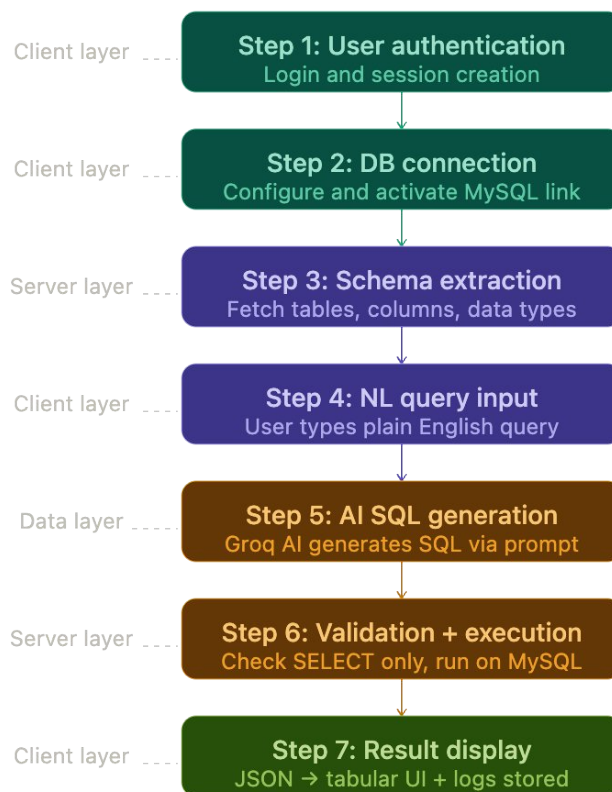

Fig. 8: End-to-End Query Processing Pipeline

Step 1 — User Authentication

The process begins with user registration and login through the React-based frontend. Upon successful credential verification by the ASP.NET Core Authentication Service, a session token is issued and stored in the client cache. All subsequent API requests carry this token to authenticate and authorize the user, ensuring that database connections and query histories remain isolated per user.

Step 2 — Database Connection and Configuration

The authenticated user provides MySQL connection parameters — host, port, username, and password — through the Configuration panel. The backend establishes a live connection to the specified MySQL instance and verifies connectivity. The system supports multiple registered connections, allowing users to switch between different databases or schemas at runtime without re-authentication.

Step 3 — Schema Extraction

Once the database connection is active, the Schema Loader module queries the MySQL information schema to extract the complete structure of the selected database — including all table names, column names, and their respective data types. This schema metadata is formatted as structured text and cached for the current session. Critically, no actual data records are retrieved or stored at this stage, preserving data privacy.

Step 4 — Natural Language Query Input

The user enters a query in plain English through the search bar on the home interface. The frontend displays table name suggestions derived from the loaded schema and maintains a recent query history from the session cache to assist the user. Upon clicking Fetch Data, the natural language query string is sent to the backend REST API as part of a structured request payload.

Step 5 — Schema-Aware AI SQL Generation

The backend constructs a structured system prompt by combining the extracted schema metadata with the user's natural language query and sends it to the Groq AI engine. The system prompt instructs the AI to generate a valid SQL SELECT statement based solely on the provided schema context, without making assumptions about data values. The Groq AI engine processes this prompt using its underlying large language model and returns a syntactically correct SQL query as a text response.

Step 6 — Query Validation and Execution

The SQL query returned by the Groq AI engine is passed through the Query Engine validation module. The validator checks that the query contains only SELECT statements and rejects any query containing data modification keywords such as INSERT, UPDATE, DELETE, DROP, or ALTER. This read-only enforcement ensures system safety and prevents unintended data modification. The validated query is then executed against the user's connected MySQL database, and the resulting rows are collected as a JSON-formatted response.

Step 7 — Result Display and Session Logging

The JSON result returned by the backend is parsed by the React frontend and rendered as a structured data table, displaying column headers, row values, total row count, and query execution latency in seconds. The generated SQL query is also made available through the View SQL Trace option for transparency. Query results can additionally be exported as CSV files. The current session's query history is maintained in the client-side cache, while all executed queries and their metadata are permanently logged in the backend database through the Logs module for audit and review purposes.

5. RESULTS AND DISCUSSION

The proposed DataExtractor system was tested across multiple custom relational database domains including e-commerce, banking, and school management. A diverse set of natural language queries were submitted to evaluate the system's ability to correctly generate and execute SQL queries. The results demonstrate that the system successfully handles a wide range of query types — from simple data retrieval to complex multi-table aggregations — provided the queries are within the scope of the connected database schema.

5.1 Query Translation Results

Table 1 presents a representative set of natural language queries tested across the three database domains, along with the type of SQL generated and the execution outcome.

Table 1: Sample Query Test Results

Natural Language Query	Database	SQL Type	Result
Show most popular product from each city	E-commerce	Multi-table JOIN + GROUP BY	Success
Total revenue of each product	E-commerce	JOIN + SUM aggregation	Success
Products added to cart but never ordered	E-commerce	LEFT JOIN + NOT EXISTS	Success
Accounts with available balance above 5000	Banking	SELECT + WHERE filter	Success
Show all data from students	School	Simple SELECT *	Success
Show yesterday's stock market prices	E-commerce	Out-of-schema query	Failed

** Failed case represents a query referencing data not present in the connected database schema.*

5.2 System Accuracy

The overall query translation accuracy of the system was observed to be in the range of 80% to 85% across all tested databases and query types. The accuracy was evaluated based on whether the generated SQL query correctly retrieved the intended data from the database. As presented in

Table 2, query accuracy varies with complexity — simple single-table retrieval queries consistently achieved high accuracy, while more complex queries involving multiple JOINs and aggregations showed moderate accuracy dependent on the clarity and precision of the user's natural language input.

Table 2: Accuracy by Query Complexity

Query Type	Example	Accuracy
Simple retrieval	Show all students	~90%
Filtered retrieval	Accounts with balance above 5000	~85%
Aggregation queries	Total revenue of each product	~78%
Multi-table JOIN queries	Most popular product from each city	~72%
Out-of-schema queries	Data not in connected database	0%

5.3 System Performance

The end-to-end response time of the system — from the submission of a natural language query to the display of results — was measured across multiple test queries under standard network conditions. Simple queries such as single-table retrievals were processed and returned within approximately 0.8 to 1.5 seconds. Complex queries involving multiple table JOINs, aggregations, and subqueries required between 2 to 5 seconds depending on the query complexity and network conditions. The system latency is primarily influenced by the round-trip time to the Groq AI engine for SQL generation, with local query execution on MySQL contributing minimally to the overall response time.

5.4 Discussion

The experimental results confirm that the proposed system effectively bridges the gap between non-technical users and relational databases. The system performs reliably for queries that are within the scope of the connected database schema and are expressed in reasonably clear English. The primary factor influencing translation accuracy is the quality and clarity of the user's natural language input — precisely phrased queries consistently yield correct SQL, while vague or ambiguous phrasing may lead to partially correct or unexpected results. This behaviour is inherent to LLM-based generation and is consistent with findings reported in recent Text-to-SQL literature.

Queries that reference data, tables, or concepts entirely outside the connected database schema are expected to fail, as the AI engine has no basis to generate a valid SQL query without schema context. This is a known and acceptable limitation — the system is designed to operate exclusively within the user's own database environment. The schema-aware prompting approach, read-only

query validation, and transparent SQL trace feature collectively make the proposed system a practical and trustworthy solution for natural language database interaction.

6. CONCLUSION

This paper presented DataExtractor, an NLP-driven Text-to-SQL system that enables non-technical users to interact with relational databases using plain English queries, eliminating the need to write or memorize complex SQL statements. The system addresses two significant barriers in database accessibility — the technical knowledge required to write SQL syntax, and the cognitive burden of remembering long, complex query structures for routine data retrieval tasks. By leveraging Groq AI-powered large language model generation with schema-aware prompting, the proposed system automatically converts natural language inputs into executable SQL queries and displays results in a clean tabular format.

The system was successfully developed and tested across multiple database domains including e-commerce, banking, and school management, achieving an overall query translation accuracy of 80% to 85% for well-formed natural language inputs. The system demonstrated consistent performance across major web browsers and operating environments. A notable feature of the proposed system is its support for multiple database connections per user — a single user can register and manage multiple MySQL server connections and switch between server instances at runtime, enabling flexible multi-database workflows without the need to re-authenticate or reconfigure the system. The schema-aware query generation approach, where only the database schema is shared with the AI engine and not the actual data records, ensures data privacy and produces contextually accurate SQL output. Additional features including read-only query validation, SQL trace transparency, interactive schema exploration, session-based query history, and CSV export collectively make the system a practical and trustworthy solution for data retrieval.

The primary limitation of the system is that translation accuracy is sensitive to the clarity and precision of the user's natural language input. Users who phrase their queries clearly and within the context of their database schema consistently achieve accurate results, while vague or ambiguous inputs may yield partially correct SQL. Additionally, the system's response time is influenced by network conditions, as SQL generation relies on an external AI engine via API. These limitations are inherent to LLM-based approaches and are well-documented in existing Text-to-SQL literature.

Future work will focus on extending support to all major SQL-based relational database systems including PostgreSQL, Microsoft SQL Server, Oracle Database, and SQLite — making the system universally applicable to any database environment that operates through structured query

language. Further improvements may include enhanced prompt engineering techniques to improve accuracy on complex multi-table queries, support for multilingual natural language inputs, and the introduction of query suggestion intelligence based on schema analysis to further assist non-technical users in formulating effective queries.

REFERENCES

[1] V. Zhong, C. Xiong, and R. Socher, “Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning,” arXiv preprint arXiv:1709.00103, 2017.

[2] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, “Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task,” in Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP), Brussels, Belgium, 2018, pp. 3911–3921.

[3] T. Yu, M. Yasunaga, K. Yang, R. Zhang, D. Wang, Z. Li, and D. Radev, “SyntaxSQLNet: Syntax Tree Networks for Complex and Cross-Domain Text-to-SQL Task,” in Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP), Brussels, Belgium, 2018, pp. 1653–1663.

[4] J. Guo, Z. Zhan, Y. Gao, Y. Xiao, J. Lou, T. Liu, and D. Zhang, “Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation,” in Proc. 57th Annual Meeting of the Association for Computational Linguistics (ACL), Florence, Italy, 2019, pp. 4524–4535.

[5] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” in Proc. Conference of the North American Chapter of the Association for Computational Linguistics (NAACL), Minneapolis, USA, 2019, pp. 4171–4186.

[6] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer,” Journal of Machine Learning Research (JMLR), vol. 21, no. 140, pp. 1–67, 2020.

[7] N. Nihalani, S. Silakari, and M. Motwani, “Natural Language Interface for Database: A Brief Review,” International Journal of Computer Science Issues (IJCSI), vol. 8, no. 2, pp. 600–608, 2011.

[8] X. Zhu, Z. Jin, Y. Liu, C. Zhang, T. Huang, and W. Hu, “Large Language Model Enhanced Text-to-SQL Generation: A Survey,” arXiv preprint arXiv:2410.06011, 2024.