AI Powered Platform for Natural Language Database Interaction and Business Intelligence Using LLM

Pradyumna Ragothaman[1] , Jayasri B S [2], and Mohammed Adnan [3]

The National Institute of Engineering
Mysuru, India
pradyumnaragothaman@gmail.com

Abstract. Structured query language (SQL) involves technical writing expertise that poses a big obstacle to non-technical users, thereby restricting data-driven decision making at the organizational level. Available solutions like direct database clients or the conventional Business Intelligence (BI) solutions are either too complicated or lack agility. To solve this, we are introducing a web-based application that will democratize data access by providing a single platform that is integrated with AI. Our system uses Large Language Models (LLMs) through a Retrieval-Augmented Generation (RAG) pipeline to generate an executable SQL query given a natural language prompt to allow users to speak to their data. The platform combines database connectivity, schema exploration, dual mode query interface, alongside automated reporting, into a single workspace. It is secure and interoperable with Enhanced Role-Based Access Control (RBAC) and API-first structure. The testing shows that the system is indeed effective in bridging the gap between complex databases and business users and thereby making the data-driven culture a reality through providing rapid and easily accessible insights.

Keywords: Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), Business Intelligence, Data Democratization.

1 Introduction

In the new data-driven environment, the fact that one can quickly access, query, and interpret information stored in databases is a key factor of any business success. Nonetheless, there still exists a major operation bottleneck, technical know-how to write Structured Query Language (SQL) is still an overwhelming force. Such dependency forms a gap between data and their decision-makers and limits the direct access to data to the small group of developers and data scientists. This means that it delays business analysts, marketers and executives in terms of innovation and agility. Although there are strong Business Intelligence (BI) applications such as Tableau and Power BI, they can be quite expensive with high licensing charges, steep learning cycles, and complicated processes, so most organizations cannot afford them, including startups working on limited applications. Hence considering the existing limitations, the proposed work

R. Vasanth Kumar Mehta et al. (eds.), *Proceedings of the International Conference on Intelligent Systems for a Sustainable Future (ISSF 2026)*, Atlantis Highlights in Intelligent Systems 16,
https://doi.org/10.2991/978-94-6239-693-7_75

was carried out at a startup known as Spense. This paper proposes an AI-based database management and analysis platform that is aimed at filling this gap. The web-based application acts as an unified workspace which pulls database connectivity, schema exploration and analytical features. Its fundamental innovation is a conversational interface, which is powered by an LLM, to query data in natural language. The system removes the barrier to SQL syntax by generating precise SQL via a Retrieval-Augmented Generation (RAG) pipeline by translating intuitive prompts into correct SQL. Use of LLM with an SQL database allows natural language integration with structured data in a faster manner and helps a non-technical user to understand the schema with democratization of data thus assisting in placing the query in conversational pattern with the help of AI powered dashboards[1].

2 Problem Statement

The current organizations are rich in data and poor in insight in that the desirable information in their relational databases is not readily available to most business users. The main barrier is the technical barrier caused by SQL which limits direct data interaction to the technical staff. This gives rise to a severe dependency, in which non-technical users, including business analysts, managers, and executives, have to depend on data teams just to make elementary data requests. Such a request-fulfilment cycle is ineffective in its own right and results in massive delays, communication, and slower decision-making, which in turn leads to the undermining of organizational agility. The current solutions could not sufficiently address this. Direct database users need extensive knowledge of SQL, and traditional BI systems are usually expensive, complex, and not suitable to ad-hoc exploration. This creates a big void in the case of an agile, intuitive and cost-effective tool that empowers everyone. The gist of it is that what is really needed is a single platform that will give the raw strength of a database client, the ease of use of a natural language interface and really democratize access to data without reducing the ability to maintain security and control.

3 Literature Survey

Collectively, the surveyed literature underpins the design of an intelligent, secure, efficient Text-to-SQL data platform aligned with the project's roadmap. Zhou et al. [2] also go beyond simple SQL synthesis by showing that LLMs can analyze EXPLAIN plans and recommend optimizations such as index creation and join reordering, thus supporting the project's ambition regarding AI-assisted performance tuning. Lin et al. [1] address hallucination by advocating rich contextual grounding and demonstrating that hybrid RAG substantially improves SQL reliability, which validates the project's choice to use a schema-metadata RAG pipeline. Kim et al. [7] emphasize rigorous evaluation through the Spider-Gen benchmark as a way to expose failure modes with nested and implicit-logic queries, providing a concrete framework for assessing the project's AI query generator. Yang et al. [3] frame core NLIDB security risks and demonstrate that AST-level policy enforcement can effectively constrain unauthorized SQL directly,

which is in support of the security-first, multi-layer validation strategy of the project. Chen et al. [6] add robustness through self-correction, an iterative, execution-feedback-based refinement that allows the system itself to debug SQL autonomously and provide a guide for evolution toward a self-improving agent.

Iyer et al. [5] show that dynamic example selection DIN-SQL is extremely effective for few-shot prompts and greatly improves Text-to-SQL performance; thus, guiding the prompt engineering design for the project. Nakamura et al. [4] describe a multi-agent, planner-based approach for decomposing complex queries into simpler sub-queries which predicts the project's Agentic LangGraph architecture for multi-step reasoning and tool use. Defog et al. [10] illustrate that fine-tuned smaller models SQL-Coder can achieve and sometimes outperform large general LLMs with less cost and latency. Thus, reinforcing the project strategy of using scalable and cost-effective specialized models. Liu et al. [12] and Wu et al. [9] emphasize the importance of explicit schema linking and context-enriched schema aware few-shot prompting for accuracy, thus supporting the project schema-focused RAG and prompting pipeline. Lastly, Ibrahim et al. [11] extend the threat model to indirectly include prompt injection and various attack vectors at the boundary between the database-LLM, thus underlining the need for strong security in depth at each stage of data flow in the project.

4 Existing System

The present database interaction and analysis environment is characterized by three major paradigms. Direct Database Clients (e.g. MySQL Workbench, DBeaver) are very powerful and have a low level access to technical people such as developers and DBAs. The conventional Business Intelligence (BI) Platforms (e.g., Tableau, Power BI) provide a sophisticated visualization and dashboarding. Lastly, SaaS applications offering Embedded Analytics Modules offer limited reporting capabilities that are contextual.

Disadvantages of current Systems:

1. High Technical Barrier: Direct database clients have very high technical requirements due to their knowledge of SQL and database structure and hence cannot be used by non-technical business users.
2. Cost Prohibitive: Conventional BI systems require substantial licensing, deployment and maintenance, rendering them impractical with smaller organizations.
3. Steep Learning Curve: BI tools are too complex and require extensive training to use their complex interfaces and data modelling.
4. Absence of Real-Time Agility: These systems are also usually not well suited to rapid, ad-hoc exploration as changing data models can be a slow and complicated operation.
5. Siloed and Limited Analysis: Embedded analytics modules are hard and specific to the application where they are deployed and do not allow them to cross-analyze databases and offer minimal customization.
6. Non-Contextual and Static: These two limits the ability to access data as they are largely automated based on the history, which is not responsive to the contextual needs of the user.

The Proposed System has the following advantages:

1. Democratized Data Access: An AI-powered natural language interface enables non-technical users to query data in a conversational fashion to overcome the SQL barrier.
2. Unified Workspace: Defines the database connectivity, schema exploration, query editing, query and report generation as well as dashboarding in a single, smooth application.
3. Cost-Effective Solution: It is based on a modern, open-source technology stack, which offers enterprise-grade functionality at a fraction of the cost of BI solutions.
4. Improved Organizational Agility: It removes technical dependencies, which significantly decreases the time to question to insight and thus makes it easier to have data-driven decision-making.
5. Active and Unscheduled: The characteristics of the platform such as scheduled reporting and API endpoints are proactive and automated to enable an active and proactive culture of data.

Strong Security and Governance: There is a centralized Role-Based Access Control (RBAC) model that is being used to provide secure, auditable, and policy-compliant access to data throughout the organization.

5 Proposed System

Proposed system is a single web application that essentially transforms the nature of interaction with the database by embedding the state-of-the-art of artificial intelligence in the data analysis process. The fundamental novelties of the system are in its conversational, agentic layer of AI, which is no longer based on simple natural language-to-SQL translation. In contrast to existing NLIDBs, This system uses Retrieval-Augmented Generation (RAG) with LangGraph to reason state fully, multi-step over a given query and the Model Context Protocol (MCP) to run tools. This enables the system to be a smart data analyst: it is able to break down complex business queries, invoke tools to examine schema and sample data, run queries in an iterative manner, and provide a holistic response- all within a single conversation.

The platform is designed to be of single, unified workspace which removes fragmentation of tools. Its salient characteristics are:

1. Single AI-Human Interface: providing both advanced SQL editor and the groundbreaking Prompt Tab query, it is a product that fulfills both the requirements of technical and business users.
2. Security-by-Design Governance: Enterprise-grade Role-Based Access Control (RBAC) is inbuilt and applying policies on the database, table, column, and row levels to all queries, either AI-generated or hand authored.
3. Proactive Analytics Engine: A Delivery and Scheduling engine is an automated system that converts ad-hoc queries into persistent and shared assets giving a scheduled report and a live dashboard that push intelligent insights directly to stakeholders.

4. API-First & Extensible Core: All reports and visualizations are exposed out of the box as a secure REST API endpoint, and This system is a central data hub that can power insights into other enterprise applications and portals.

This system can not just give people access to data--by combining a smart AI agent with a safe multi-tenant platform with a full analytics life cycle in a single app, it will allow an organization to begin to transition to pervasive, self-service, and data-driven decision-making.

6 System Design

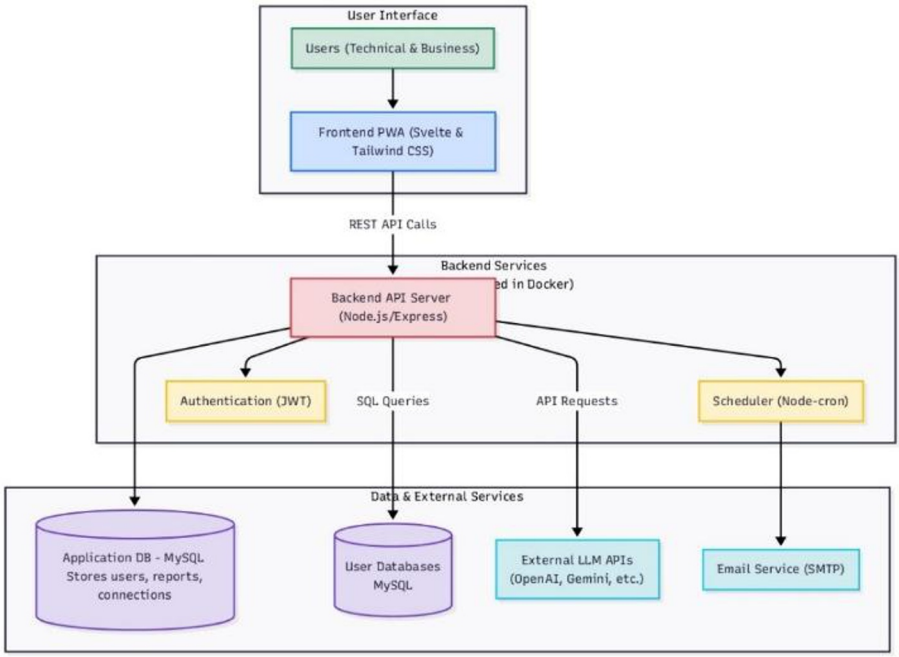

Fig. 1. System Architecture

The system is designed as a modern three-tier architecture with a clean separation of concerns to guarantee scalability, maintainability and security. The high-level System Architecture is as shown in "Fig.1", which divides the application into a Presentation Layer, Application Layer, and a Data & External Services Layer. This hierarchy enables the autonomous growth and expansion of every part and offers a solid base of the sophisticated functions of the system.

Presentation Layer The client-side Progressive Web Application (PWA) is composed of Svelte and Tailwind CSS. The layer serves as the main interface with two different user personas, namely: Technical User (e.g., data analyst) and the Business User (e.g., marketing manager). It is stateless, and its only interaction with the backend is through secure HTTPS REST API calls and it is decoupled and responsive to the

user. It has a single workstation, which is an all-encompassing platform that combines all the features, including database management and visualization of the dashboard, without context-switching between separate tools.

The Business logic of the system is implemented in the Application Layer that is a container-ready Node.js server running on the Express.js framework. It is stateless to enable horizontal scaling. This layer coordinates all the operations of the back end, such as user authentication with the help of JSON Web Tokens (JWT), talking with a database, and connecting with third-party AI services. It supports multiple central modules: the Authentication module to have secure access; the Data Connection Manager to operate encrypted database credentials; the Schema Module to provide metadata; the Query Module to execute secure SQLs, the AI/Prompt Module which includes the RAG pipeline and the Report & Scheduler Module to manage analytical resources and deliver them in an automated way. Data and External Services Layer contains all stores of persistence and external APIs. Logically it is split into the Application Database and User Databases. All metadata of the platform such as user profiles, connection details, report definitions, and audit logs are stored in the Application Database (MySQL). The external MySQL databases with which users can be connected to analyse themselves are the User Databases; the platform serves as a client to these databases. The Milvus vector database used to store schema embeddings and the external LLM APIs (e.g. OpenAI, Gemini) to process natural language are also available in this layer.

One system innovation that is essential and critical is the AI-driven Conversational Query Interface, also referred to as Prompt Tab. It starts when a user enters a natural language query. The system loads and vectors the schema metadata (tables, columns, data types) of the connected database into Milvus. It does a similarity search at query time to bring out the most relevant schema elements. This context is forcibly fed into a well-designed prompt template and inputted to a Large Language Model (LLM) which produces a syntactically and contextually appropriate SQL query.

Fig. 2. Agentic AI Data Flow MCP and LangGraph Orchestration.

The system uses a high-level AI orchestration framework in the case of complex and multi-step analytical questions. With the help of LangGraph, the AI is a stateful agent capable of reasoning and acting as depicted in the Multi-step Analytical Query Execution Diagram as shown in "Fig.2". It is based on ReAct (Reason-Act) pattern, which breaks down a complex request of a user into smaller sub-tasks. It then applies to a set of tools which are standardized through the Model Context Protocol (MCP) to carry out actions such as sampling data or controlled queries.

The state of the agent, conversation history, and the results of the tool are checkpointed, allowing really conversational and multi-turn data exploration. The sequence of the delivery of such insights is presented in the Sequence Diagram of Loading a Dashboard (Fig. 3) which indicates the optimization of performance of the system. In this case, various report widgets on a dashboard retrieve their information using asynchronous and parallel API calls, and provide the end-user with a fast and responsive rendering experience. This is a unified design, not only of the overall architecture, but also of particular user interfaces to AI and dashboards, which makes the platform mighty, safe, and easy to use.

7 System Implementation

The evolution of the AI-Integrated Database Management and Analysis Platform was done in a systematic, five-phase approach to the logical transformation of architectural designs into a working application. One stage was based on the completion of the previous stage, and it guaranteed the gradual provision of functionality without compromising the code quality and integrity of the system.

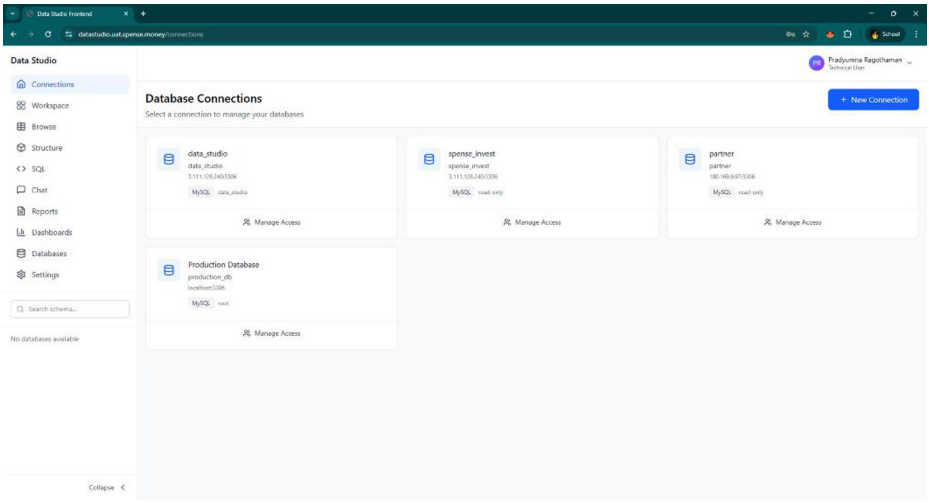

Fig. 3. DB interface with credential secure storage.

Phase 1: Foundation and Core Infrastructure. The first stage formed the basic pillars of the platform. Our initial setup was a scaffold of the project, that is, two independent

frontend (SvelteKit) and backend (Node.js/Express) directories. The application database schema was created and configured in MySQL where it created user management, database connections, report and audit tables. JWT (JSON Web Tokens) and bcrypt.js were used to create the authentication module, allowing users to be registered, log in, and manage the session through the secure password hashing. One of the most important parts of this stage was the Database Connection Manager (DBCM) which enables users to add and store credentials to external MySQL databases as shown in "Fig.3". Sensitive credentials are encrypted with AES-256 and then stored in dim_connections table.

Phase 2: Data Interaction Layer. This step was aimed to allow direct access to user databases. We used the Schema Module that offers REST endpoints to retrieve and present metadata of databases in a tree format. The Query Module was created to support secure execution of SQL queries over the /api/query/execute endpoint with query validation, protection against timeouts and adequate error handling. The user interface of this stage consisted of an advanced SQL editor that has syntax highlighting and autocompletion and a schema explorer panel.

```
1. def generate SQL prompt(userprompt, connectionid):
 2. In the first step, the context of relevant schema is
retrieved.
 3. schemacontext = vectordb.similaritysearch(userprompt,
connectionid)
 4. Step 2: Develop improved prompt.
 5. enhancedprompt = f""" Database Schema Context: {sche-
macontext} User Question: {userprompt} Generate SQL
Query: """
 6. # Step 3: Call LLM
 7. generatedsql = generatedsql = lllmapi.generate(en-
hancedprompt)
 8. # Step 4:
 9. Validate and return validatesql(generatedsql)
```

Fig. 4. Streamlined RAG Pipeline Pseudocode.

Phase 3: AI-Based Intelligence Integration. The third step ushered in the most innovative feature of the platform, which is the natural language to SQL conversion. We used a Retrieval-Augmented Generation (RAG) pipeline, which starts with schema ingestion and vectorization. In a connection to a new database, metadata (names of tables, columns, types of data) is processed with an embedding model (Azure OpenAI embeddings) and stored in a Milvus vector database. AI query endpoint receives user prompts, which are then processed by making similarity searches in the vector database, retrieving relevant schema context, and generating further prompted prompts, to be processed by LLM (OpenAI GPT-4 or Google Gemini) as shown in "Fig.4". The same is depicted in sequence diagram in "Fig.5". Before running the generated SQL, it is validated.

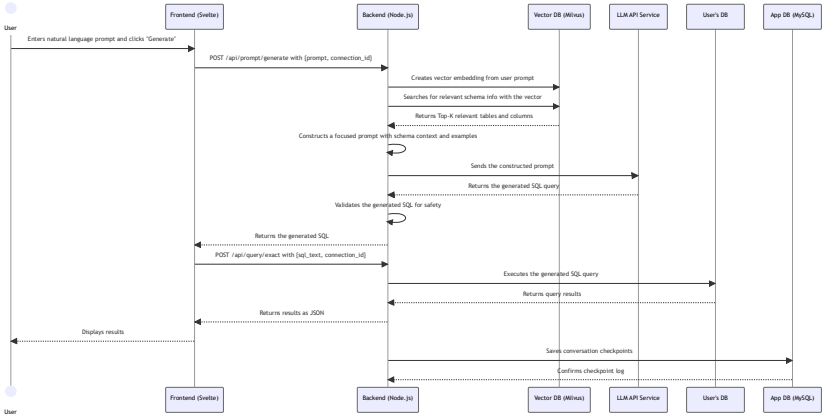

Fig. 5. Sequence Diagram - AI-Powered Query

Phase 3: AI-Based Intelligence Integration. The third step ushered in the most innovative feature of the platform, which is the natural language to SQL conversion. We used a Retrieval-Augmented Generation (RAG) pipeline, which starts with schema ingestion and vectorization. In a connection to a new database, metadata (names of tables, columns, types of data) is processed with an embedding model (Azure OpenAI embeddings) and stored in a Milvus vector database. AI query endpoint receives user prompts, which are then processed by making similarity searches in the vector database, retrieving relevant schema context, and generating further prompted prompts, to be processed by LLM (OpenAI GPT-4 or Google Gemini) as shown in "Fig.4". The same is depicted in sequence diagram in "Fig.5". Before running the generated SQL, it is validated.

Phase 4: Reporting, Dashboarding and Automation. Phase 4 changed the platform to a query tool to business intelligence solution. Report Module allowed users to save queries in the form of parameterized reports. The Dashboard Module offered a drag-and-drop interface when it comes to the creation of multi-widget dashboards where data is loaded asynchronously and in parallel. The Scheduling Engine was created using cron that enabled users to create automated report generation and delivery through email or built-in systems. It was also a time when API-first design concepts were introduced and all reports were presented as safe REST endpoints.

Phase 5: Security Baselining and Deployment. The last stage was on readiness to production. We provided extensive security practices such as end-to-end encryption of the API with a custom secureApi client, intense enforcement of Role-Based Access

Control (RBAC) at both database and table and row level and prevention of SQL injections by using parameterized queries.

Docker was used to containerize the application, with frontend, backend, MySQL, Redis and Milvus services being in separate containers. All services were coordinated by a production docker-compose.yml file and the platform was released to a cloud service (AWS EC2) including monitoring and logging services and a backup system.

8 Testing

The functionality, security, and performance of the platform were tested in a comprehensive, multi-layered strategy to ensure that the platform can be used in production. The testing methodology as shown in “Fig.6”, followed the testing pyramid with high emphasis on good base of unit tests, which was complemented with integration, system and user acceptance testing. All core features were tested to be functional. The authentication module was strictly tested to secure login, JWT-token handling, as well as a strict implementation of Role-Based Access Control (RBAC), which ensured that users accessed a database or a functionality that they should have accessed. The Data Connection Management (DCM) was observed to have a full lifecycle, that is, its ability to add and test new connections including encrypted credential storage and to edit and revocation of access. The schema exploration and query execution modules were tried to have a proper metadata retrieval and safe execution of different SQL statements, including syntax errors and query timeouts. The AI was validated in a particular way, a series of natural language prompts was filtered and then evaluated against the accuracy, syntactic correctness, and semantic validity of the generated SQLand is queries (see Table. 1).

The most important thing was security testing. We also performed penetration tests on every API endpoint and managed to check the system with regard to its resiliency to common vulnerabilities, such as SQL injections and cross-site scripting (XSS). Enactment of row-level security policies was also confirmed as effective and only allowed the user to access information that they were allowed to access according to their role. The performance and load test allowed determining that the system was within specifications and the queries with less than 10,000 rows could be executed within less than two seconds, and the dashboards with various widgets could be loaded effectively through parallel data retrieval. Lastly, User Acceptance Testing (UAT) by target end users ensured that the platform was usable and could pass real-world business analysis processes and the time-to-insight was greatly reduced than the traditional process.

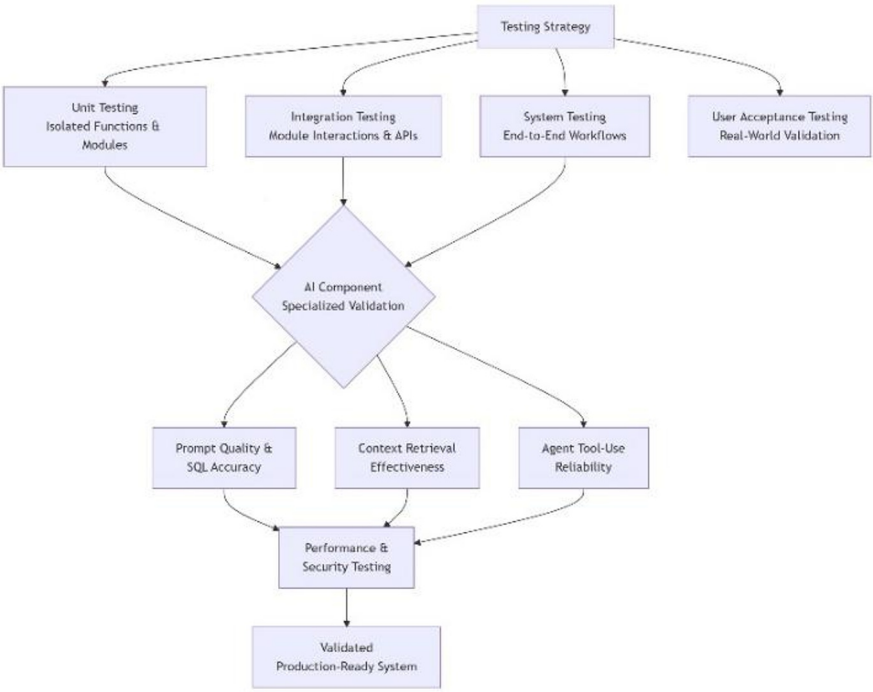

Fig. 6. Strategy of comprehensive testing

Table 1. Test Case for testing all the functional requirements

Test Case ID	Test Scenario	Status	Result Summary
AI-01	Simple NL-to-SQL: "Top 10 customers by purchases"	PASS	Correct SQL generated and run.
AI-02	Complex query: Revenue by product category last quarter	FAIL	Interpretation of last quarter as ambiguous resulted in omission of date filter.
AI-03	Agentic multi-step query: Compare top 3 category sales between quarters	PASS	Agent decomposed query and performed stepwise correctly

SEC-01	SQL injection attempt through prompt	PASS	Malicious DROP command prevented by validation layer
SEC-02	Enforcement of row level security	PASS	Only user could view information within his/her authorized area
SEC-03	5 Widgets loaded on Dashboard	PASS	All Widgets loaded in parallel (Less than 4 seconds)
SEC-04	Self-service workflow of business user	PASS	Insight of business user through natural language without SQL

9 Results And Discussion

The testing and implementing this platform confirmed its main goals with good quantitative and qualitative outcomes. The system met its functional performance objectives and the queries yielding less than 10,000 rows took less than 2 seconds to finish.

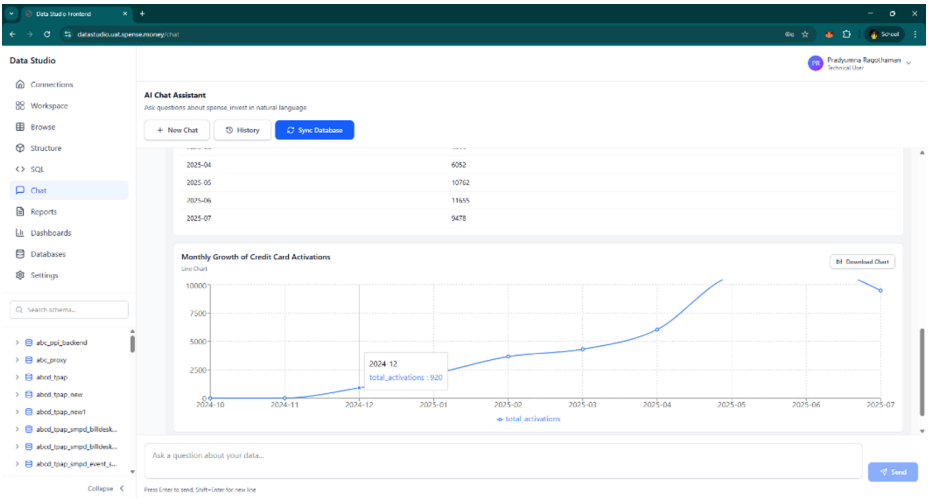

Fig. 7. AI-Powered 'Prompt' Tab Interface

The query generation AI showed 89 percent accuracy rate on a wide range of test cases. It was essential to integrate a Retrieval-Augmented Generation (RAG) pipeline in order to cut the number of schema hallucinations by about 73%. More importantly, the agentic reasoning layer which is coordinated through LangGraph, was able to break down 82% of multi-step, complex analytic queries into legitimate sequences of tool interactions and facilitated conversational-level data exploration as shown in “Fig.7”. The defense-in-depth approach was verified to be very strong in security testing through 100 percent of SQL injection attacks blocked and Role-Based Access Control (RBAC) enforcing all the data access policies correctly.

User Acceptance Testing indicated that about 67% drop in time-to-insight was achieved with the business users, who were able to gain access to answers using natural

language and without having to know SQL. The integrated workspace was appreciated by technical users because they could no longer have to switch contexts between dissimilar tools. When compared to existing method that limits non-technical user from accessing database with direct database client, BI tools, embedded analytical modules that results in delayed decision-making missing opportunities of growth in business. These findings show that the platform has managed to close the gap between power generation of data access and usability by the user. It is technically viable, and can be demonstrated to add tangible value to the creation of a stateful AI agent, a secure multi-tenant architecture, and a complete analytics lifecycle into one application.

Nevertheless, it was found that testing revealed major limitations which were the ambiguity in making judgements regarding relative time references (e.g., last quarter) and a trade-off where complex queries with the advanced agentic capabilities added slightly higher latency than the base RAG pipeline. The specified shortcomings offer a good map to follow in the further improvements aimed at the contextual clarification and performance optimization.

10 Conclusion

Through this project, a working, validated and designed AI-based database management and analysis platform has been developed that intelligently addresses the challenge of connecting the complex data infrastructure to non-technical users. The proposed system addresses the limitations of traditional database clients, BI tools, and embedded analytics platforms by introducing an AI-driven, unified data interaction environment. Advantage of using a modern web-based architecture and an advanced RAG pipeline enables the system to automatically convert natural language prompts to correct SQL and democratizes data access and empowers users on either side of the technical spectrum. The introduction of agentic reasoning using LangGraph and MCP brings the system to the next level of a query translator, and it is capable of performing complex, multi-step analytical problems. The platform is a feasible and economical alternative to traditional BI tools, with solid security provided by RBAC, a single environment to analyze all its data, and performance metrics that prove successful, creating a culture centered on data in an organization. Overall, the solution transforms traditional data analysis workflows into an intelligent, accessible, and scalable ecosystem that empowers organizations to adopt pervasive self-service analytics.

Enhancements that will be made in future are the expansion of the capabilities and intelligence of the platform. One of the objectives is Multi-Database Support, which allows connecting not only with MySQL but also with PostgreSQL, SQL Server and cloud data warehouses such as Snowflake and Big Query. Further AIs will be developed, such as more agentic processes to identify and profile automated data, detect anomalies, and predictive analytics. In order to make the tool a collaborative center, we are going to introduce Collaborative Features, i.e., shared workspaces and comment threads on reports and dashboards. Moreover, research will center on the Improved Natural Language Understanding by finetuning models of domain-specific jargon, and

Mobile Optimization that will enable insights to be available everywhere. All these improvements will bring the limits of customized, smart data engagement.

References

1. Lin, J., Mukherjee, P., Alvarez, D.: Advanced RAG Techniques for Domain-Specific SQL Generation. ACM Transactions on Database Systems (2025)
2. Zhou, K., Desai, M., Nguyen, T.: LLM-Powered SQL Query Optimization and Cost Prediction. IEEE Transactions on Knowledge and Data Engineering (2025)
3. Yang, R., Patel, S., Torres, L.: Securing NLIDBs: Mitigating Prompt Injection via Query Skeleton Validation. In: Proceedings of the ACM Conference on Computer and Communications Security (2024)
4. Nakamura, A., Banerjee, C., Robinson, E.: A Multi-Agent Framework for Complex Analytical Query Decomposition. In: Advances in Neural Information Processing Systems (NeurIPS) (2024)
5. Iyer, D., Chen, N., Hassan, R.: DIN-SQL: A Dataset and Model for Dynamic In-Context Learning for Text-to-SQL. In: Proceedings of the AAAI Conference on Artificial Intelligence (2024)
6. Chen, T., Liu, Y., Wang, J.: Self-Correcting LLMs for Text-to-SQL Generation. Proceedings of the VLDB Endowment (2024)
7. Kim, H., Lee, S., Park, B.: Evaluating Large Language Models on Complex Text-to-SQL: The Spider-Gen Benchmark. In: Proceedings of the International Conference on Learning Representations (ICLR) (2024)
8. Singh, V., Chen, L., Kumar, D.: Enhancing Text-to-SQL with Knowledge Graph-based Retrieval. In: Proceedings of the International Conference on Management of Data (SIGMOD) (2024)
9. Wu, L., Gupta, H., Singh, A.: SQL-PaLM: Improved Large Language Model Adaptation for Text-to-SQL. Google Research (2023)
10. Defog, E., Mendez, J., Rao, K.: SQL-Coder: A Specialized Language Model for SQL Generation. Defog Inc. (2023)
11. Ibrahim, M., Zhang, T., Krishnan, S.: On the Security of Natural Language Interfaces to Databases. In: Proceedings of the IEEE Symposium on Security and Privacy (2023)
12. Liu, S., Martin, F., Gomez, R.: Schema-Aware Transformers for Text-to-SQL Generation. In: Proceedings of the Association for Computational Linguistics (ACL) (2023)