USC University of Southern California

Viterbi School of Engineering

Course: DSCI 551 – Foundations of Data Management

Project Title: ChatDB – Natural Language Interface for Databases

Author**:** Yuchen Zhu

Group: ChatDB 76

USC ID: 2244311522

Instructor: Wensheng Wu

Date of Report: May 9, 2025

1. Introduction

In recent years, the integration of natural language processing (NLP) with relational databases has opened new possibilities for non-technical users to interact with structured data. Traditional SQL interfaces, while powerful, require users to understand specific database schemas and syntax. This creates a barrier for analysts, stakeholders, and decision-makers who lack technical expertise.

To address this issue, we developed **ChatDB**, a lightweight natural language interface for MySQL-based relational databases. ChatDB allows users to ask questions in plain English and translates them into executable SQL queries using the OpenAI GPT-3.5-Turbo API. The system dynamically extracts the target database schema and includes it in the prompt to generate context-aware, accurate, and syntactically correct SQL statements.

ChatDB is implemented as a web application using Flask, supporting both **data retrieval** and **data modification** (i.e., SELECT, INSERT, UPDATE, DELETE) operations. It is compatible with multiple database schemas and demonstrates generalizability across different domains, including clinical records and banking transactions.

This project explores the practical feasibility of large language models in translating real-world natural language into actionable database queries, aiming to improve accessibility, usability, and productivity for a wide range of users.

2. Planned Implementation (From Project Proposal)

The original plan for this project, as described in the proposal, was to build a Natural Language Interface (NLI) to a relational database system using MySQL. The goal was to enable users to interact with structured data by typing natural language queries instead of manually writing SQL.

The planned functionalities included:

- **Schema Exploration**: Users should be able to ask about table structures, attributes, and sample records.
- **Query Execution**: The system would convert natural language inputs into SQL SELECT statements to retrieve data.
- **Data Modification**: The NLI was intended to support INSERT, UPDATE, and DELETE operations through natural commands.
- **Multi-Table Queries**: It would support JOINs, filtering, and aggregation operations.
- **Integration with LLMs**: OpenAI's GPT model (initially GPT-4 was proposed) would be used for interpreting queries and generating SQL.

The system was to be developed using the following technology stack:

- **Backend**: Python (Flask or FastAPI)
- **Database**: MySQL

- **Natural Language Processing**: OpenAI GPT API
- **Query Conversion**: Prompt engineering and schema-guided generation
- **Security**: Basic SQL injection prevention and possibly user authentication

The proposed workflow was:

1. The user types a natural language query into a web form.
2. The query is passed to an LLM (GPT) to generate a corresponding SQL query.
3. The generated SQL query is executed on the database.
4. The query result is returned and rendered to the user in a readable format.

This initial implementation plan provided a strong foundation for building a generalizable, domain-independent natural language interface for relational databases.

3. Architecture Design (Flow Diagram and its Description)

System Overview

The architecture of ChatDB is designed to enable seamless interaction between users and relational databases through natural language. It integrates a lightweight web interface, a backend processing layer powered by Flask and OpenAI's GPT-3.5 Turbo, and a MySQL database engine. The system is modular, scalable, and supports multiple databases with different schemas.

Flow Diagram Description

User Input (Natural Language)
↓
Web Interface (HTML + JavaScript)
↓
Flask Backend Server (Python)
↓
Schema Extractor → Prompt Constructor
↓
OpenAI GPT-3.5 Turbo API (LLM)
↓
Generated SQL Statement (SELECT/INSERT/UPDATE/DELETE)
↓
SQL Executor (SQLAlchemy + Pandas)
↓
MySQL Database (bankdb / clinicdb)
↓
Query Result or Status Message (DataFrame or Success)
↓
Web Frontend Display (HTML Table or Text)

Component Descriptions

- **Web Interface**: A simple HTML + CSS + JavaScript front-end that allows users to input natural language queries and select the target database.
- **Flask Backend**: Acts as the main logic layer, receiving input, routing requests, assembling prompts, and sending queries to the OpenAI API.
- **Schema Extractor**: Dynamically retrieves table and column information from the selected MySQL database to be included in the prompt.
- **Prompt Constructor**: Embeds schema + user question into a structured prompt to help the LLM generate accurate SQL.
- **OpenAI GPT API**: Uses `gpt-3.5-turbo` to convert the prompt into an executable SQL query.
- **SQL Executor**: Uses SQLAlchemy and Pandas to run the SQL and return either a DataFrame (for SELECT/SHOW) or a status message (for INSERT/UPDATE/DELETE).
- **Database**: Local MySQL databases (`clinicdb`, `bankdb`) contain different tables. All are schema-agnostic to the LLM as long as prompt is accurate.
- **Output Renderer**: Translates DataFrame results into HTML tables or simple messages for the frontend.

Multi-Database Support

The system supports multiple databases by allowing users to select from a dropdown menu in the UI. Based on the selection, the corresponding schema is fetched dynamically and passed into the prompt sent to the LLM. This ensures that GPT understands the structure of the active database before generating the SQL query. Switching between `clinicdb` and `bankdb` requires no server restart and is handled entirely in the frontend and Flask routing logic.

Error Handling

- If SQL generation fails, the system catches the exception and displays a user-friendly message.
- If SQL execution fails (e.g., due to syntax or constraint errors), the error is returned as feedback in the web interface.
- Prompt injection and security issues are mitigated by only sending schema and user input through a controlled format.

4. Implementation

4.1 Functionalities

ChatDB successfully implements a full natural language interface to relational databases with the following key features:

- **Natural Language Query Parsing**: Accepts English questions such as "List all patients" or "Show customers with more than 1 account" and translates them into valid SQL queries.
- **Support for Full SQL Spectrum**: Allows not only `SELECT` statements but also `INSERT`, `UPDATE`, `DELETE`, `SHOW TABLES`, and `DESCRIBE` commands.
- **Multi-Database Support**: Supports dynamic switching between `clinicdb` and `bankdb` without restarting the application. Each schema is parsed in real time to guide the LLM's SQL generation.
- **Dynamic Prompt Construction**: Uses table-column schema extraction to guide OpenAI GPT-3.5 in generating syntactically correct SQL, even for aggregation and joins.

- **Accurate SQL Execution**: Valid SQL is executed using SQLAlchemy and returned to the frontend using Pandas (for tabular results) or HTML (for success messages).

- **Web-Based UI**: A clean, minimal HTML/CSS frontend allows users to type questions, choose the database, and view results in a readable table format.

- **Basic Error Handling**: Returns error messages for failed queries or connection issues, improving usability and debug-ability.

4.2 Technology Stack

Layer	Tools Used
Frontend	HTML5, CSS3, JavaScript
Backend	Python (Flask), SQLAlchemy, Pandas, PyMySQL
LLM	OpenAI GPT-3.5-Turbo API (`openai>=1.0.0`)
Database	MySQL with two manually created databases (clinicdb, bankdb)
Environment	Localhost deployment, Google Chrome, VS Code

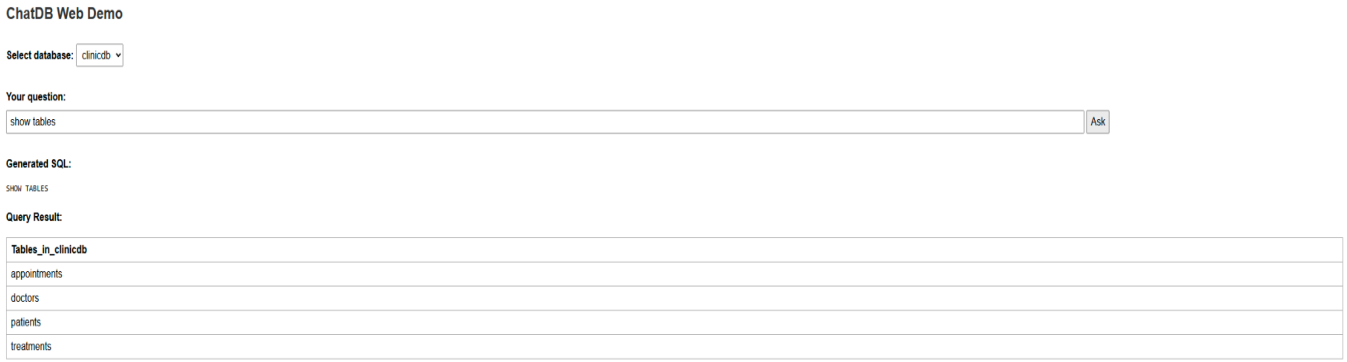
4.3

4.3 Implementation Screenshots

Figure 1. ChatDB Web Interface

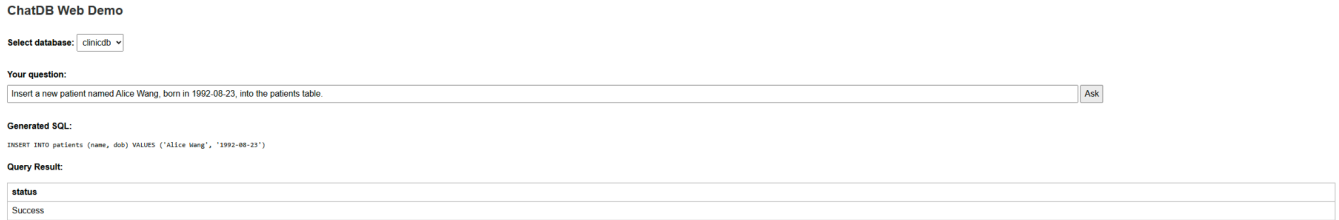
The home page of ChatDB allows users to select a database (`clinicdb` or `bankdb`) and enter a natural language question. The query will be interpreted and executed accordingly.

Figure 2. Displaying Available Tables



This allows users to quickly inspect the database structure.

Figure 3. Data Modification – Insert

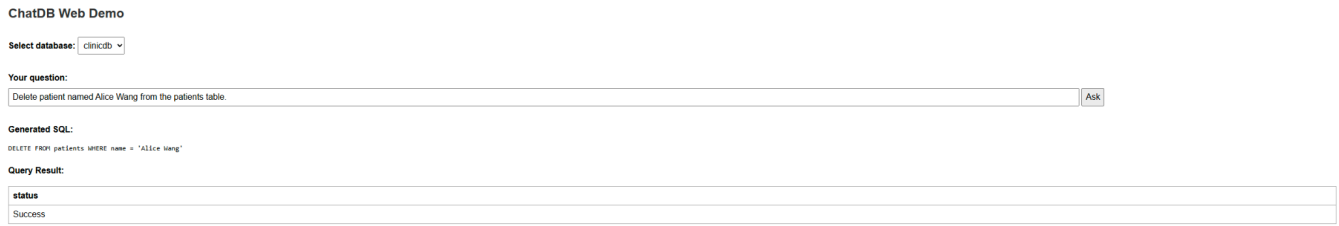


The user enters a natural language request to insert a new patient named Alice Wang born in 1992. The system correctly interprets the request and generates the corresponding SQL `INSERT` statement, executing it successfully.

Figure 4. Data Modification – Update

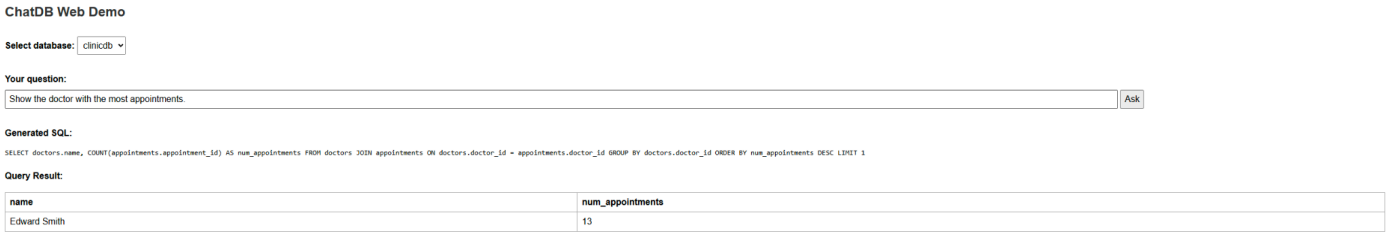
The user updates Alice Wang's date of birth to 2001-05-05 using natural language. The system parses the query and generates the appropriate `UPDATE` SQL, confirming the modification with a success message.

Figure 5. Data Modification –Delete



The user deletes Alice Wang from the database by issuing a natural language request. The system converts this into a `DELETE` SQL command and removes the record successfully.

Figure 6. Join, Aggregation & Condition Query Execution



This screenshot illustrates ChatDB's ability to process advanced SQL queries involving `JOIN`, `GROUP BY`, `ORDER BY`, and `LIMIT`.

5. Learning Outcomes

Through the implementation of the ChatDB project, I have gained valuable experience in both database systems and full-stack development. The key learning outcomes are as follows:

- **Natural Language Processing Integration**:
  I learned how to convert natural language queries into structured SQL using OpenAI's GPT model, including prompt design, schema injection, and query correction logic.

- **SQL Schema Design and MySQL Proficiency**:
  I enhanced my understanding of relational database modeling by creating two realistic databases (`clinicdb`, `bankdb`) with foreign key relationships and proper normalization. I also practiced writing and validating complex SQL statements involving `JOIN`, `GROUP BY`, `HAVING`, and conditional filtering.

- **Backend Development with Flask**:
  I gained experience building a lightweight API server that handles dynamic requests, parses input, connects with OpenAI's API, and interacts with a MySQL backend using SQLAlchemy and Pandas.

- **Data Modification with SQL Execution**:
  I implemented logic to handle not only SELECT queries but also `INSERT`, `UPDATE`, and `DELETE`, ensuring the database reflects real-time changes based on natural language commands.

- **Frontend-Backend Integration**:
  I developed a simple web interface that accepts input queries, interacts with the backend through form submissions, and displays dynamic result tables using HTML and CSS. This strengthened my understanding of full-stack data applications.

- **Error Handling and Debugging**:
  I implemented structured error messages and handling logic to support malformed queries, API failures, and schema loading issues.

These skills have prepared me to tackle larger database-driven applications and to build intelligent query interfaces in future projects.

6. Challenges Faced

During the development of ChatDB, I encountered several technical and design-related challenges:

- **1. Converting Ambiguous Natural Language into Valid SQL**
  One major challenge was that natural language queries can be vague, incomplete, or expressed in unexpected ways. Initially, the system frequently generated malformed or semantically incorrect SQL.
  Solution: I improved prompt engineering to include clear role definitions (e.g., "You are a MySQL query generator") and appended dynamic schema context. I also added pattern-based hardcoded fallbacks for common questions.

- **2. Executing INSERT/UPDATE/DELETE Safely**
  Unlike `SELECT` queries, `INSERT`, `UPDATE`, and `DELETE` statements can affect real data. At first, executing such operations blindly posed risks like integrity violations (e.g., primary key conflicts).
  Solution: I added logic in the backend (`sql_reader.py`) to distinguish between `SELECT` and data-modifying queries. `INSERT` operations were tested using auto-increment keys and verified with table inspections.

- **3. Auto-increment ID Gaps after DELETE**
  After deleting records, inserting new data continued from the last highest ID, leading to gaps in the ID column.
  Solution: This behavior was accepted as a natural aspect of MySQL's auto-increment mechanism. For presentation purposes, the focus was placed on data correctness rather than ID continuity.

- **4. Schema Extraction Compatibility**
  Reading schema metadata dynamically and formatting it for LLMs required parsing nested dictionaries and handling edge cases (e.g., tables with no columns or restricted access).
  Solution: I built a helper function `get_schema_dict()` and ensured all tables had consistent structures with meaningful names.

- **5. User Interface Formatting**
  Initially, long queries overflowed the input box, and the output tables were not properly aligned.
  Solution: I adjusted the frontend layout with CSS and HTML changes to ensure readability. The SQL and results now appear clearly under each user query.

- **6. Handling API Key Security**
  Since OpenAI's key is required, hardcoding it posed a security risk.
  Solution: The key is now loaded securely from the system environment (`OPENAI_API_KEY`), and I ensured the key is excluded from version control.

7. Conclusion

ChatDB demonstrates the feasibility and practicality of integrating natural language processing with relational databases. By leveraging OpenAI's GPT model and dynamic schema injection, the system enables users to interact with MySQL databases through plain English — without needing any knowledge of SQL syntax.

The system was designed to support both **read-only** (`SELECT`) and **data-modifying** (`INSERT`, `UPDATE`, `DELETE`) operations. I implemented safe execution layers, dynamic schema parsing, and a responsive web frontend to allow intuitive interaction. The support for multiple databases (e.g., `clinicdb`, `bankdb`) shows that the system is generalizable and extensible to other structured datasets.

Through this project, I gained deeper understanding of:

- How to apply prompt engineering techniques for reliable LLM output

- How to structure database queries programmatically
- How to build a web-based interface (Flask + HTML/CSS) that bridges AI and databases
- How to safely handle write operations with potential side effects

Overall, ChatDB proves that **LLM-assisted database querying** is not only possible but highly user-friendly. It lowers the barrier for non-technical users to access and manipulate data, offering promising directions for real-world enterprise applications and future research in database interfaces.

8. Future Scope

While ChatDB currently supports basic querying and data modification through natural language, there are several avenues for future enhancement and expansion:

1. **Advanced Query Understanding**
   The current system handles most single-statement queries well but may struggle with multi-step or nested logical conditions. Future work could incorporate **semantic parsing** or **multi-turn dialogue memory** to improve complex query handling.

2. **Multi-table Joins with Schema Reasoning**
   Although basic joins are supported, enabling the system to **automatically reason about foreign keys** and suggest joins across large schemas would improve usability on enterprise-grade databases.

3. **User Authentication & Permissions**
   Adding user-level authentication would allow access control over certain tables or operations, which is crucial in production environments such as medical or financial systems.

4. **Natural Language to Visualization**
   Future versions could allow natural language inputs like *"show me a bar chart of monthly appointments"*, which the system could convert into both SQL and a visualization using libraries like **Plotly** or **Matplotlib**.

5. **Support for Other DBMSs**
   Extending support from MySQL to **PostgreSQL**, **SQLite**, or **NoSQL** backends (e.g., MongoDB) would broaden the use cases and applicability of the system.

6. **LLM Output Verification**
   Implementing a **SQL validator or fine-tuned LLM** to double-check generated SQL can help avoid invalid or unsafe queries, increasing overall robustness and user trust.

7. **Deployment and Scalability**
   Packaging the project into a **Docker container**, and deploying it on cloud platforms (like Heroku or AWS), could allow the system to serve real users and support scalability testing.

By addressing these areas, ChatDB can evolve into a robust, production-ready natural language interface system for databases that bridges the gap between human intuition and structured data logic.

9. Future Scope

📂 Project GitHub Repository

https://github.com/Yuchen-7/ChatDB

📄 Final Report (Google Docs)

https://docs.google.com/document/d/19iqLVAFhkvHeIlHnE5lFUKjq8v0kstXsjKvQyD6erss/edit?usp=sharing

(Access: Anyone with the link – View only)