University of Southern California

Viterbi School of Engineering
Thomas Lord Department of Computer Science

DSCI 551: Foundations of Data Management

Instructor: Prof. Wensheng Wu

Final Project:

ChatDB- Natural Language to SQL

Name: Ansh Krishna

USC ID: 6687507985

Introduction

ChatDB is an interactive natural language interface for querying structured relational databases. Its core functionality is to accept user input in plain English and return relevant database results by converting the input into SQL using a large language model (LLM). The project leverages Google Gemini, hosted via the google.generativeai SDK, to perform the natural language to SQL transformation. SQL queries are executed on a MySQL database managed locally through MySQL Workbench. The results are displayed through an intuitive Streamlit-based frontend. The goal is to enable users with no SQL knowledge to explore and interact with relational data efficiently.

Planned implementation

The project required us to create a database management system which can be managed by a natural language interface. With this interface, we can execute SQL commands using human languages such as normal conversational English sentences and the interface would convert it to SQL commands and apply those command to database.

The project required coding languages, such as Python, for backend implementations. Using an LLM API is another important component in the project which is integral for conversion from Natural Language to SQL queries. For implementation for SQL, MySQL will be used.

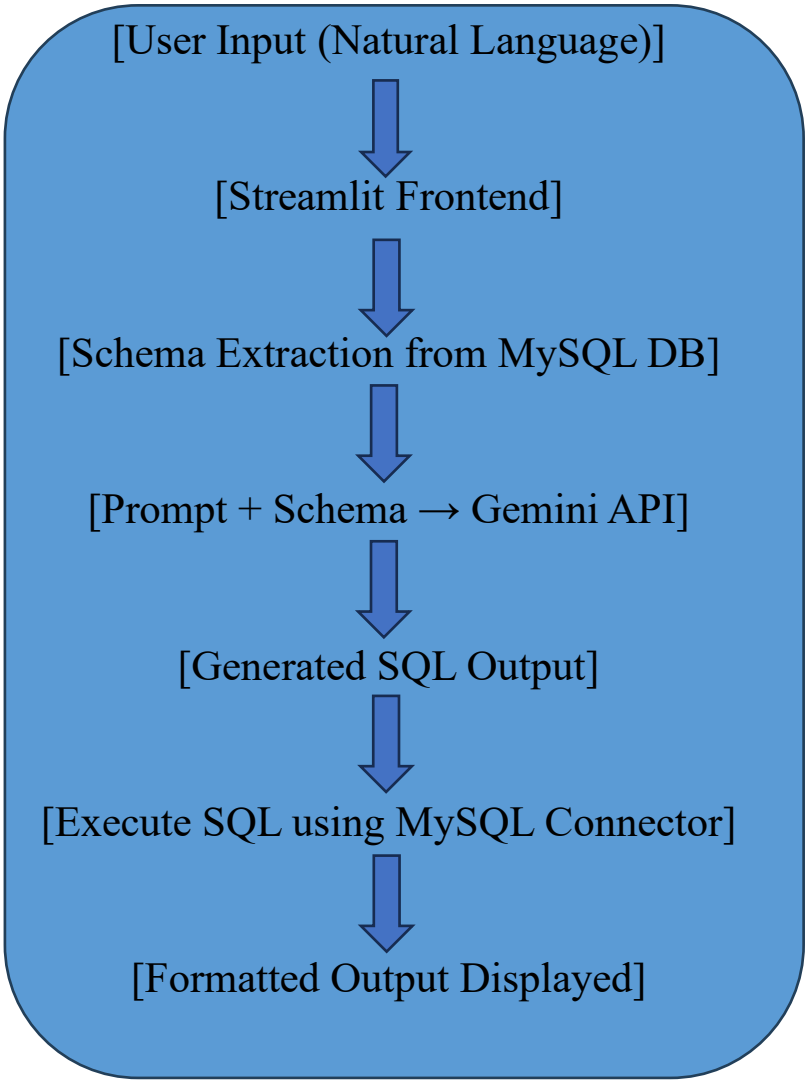
The project needs to begin with finding real world databases, which will be the main base. It defines the whole direction and gives a specified area. This process will require some research work and going through directories containing various datasets.

After this, the implementation involves Natural Language processing to extract keywords like table names, attributes, and conditions and using LLMs to fine-tune or use API for text-to-SQL. This will be followed by Text-to-SQL models to generate queries. These models can be implemented using python coding

Timeline

SR. NO	TASKS	MILESTONES/CHECKPOINTS	DURATION
1.	Project Proposal	Submitted ✅	Week 1
2.	Finding Databases and setting MySQL	Databases found and MySQL set up for use ✅	Week 2
3.	Configuring LLM API	Gemini API configured ✅	Week 3
4.	Mid-term progress report	Submitted ✅	Week 4
5.	Developing NLP module for query processing	Using LLM API to test for SQL queries and table generation ✅	Week 5
6.	Implementing SQL query generation & execution	INSERT, UPDATE, DELETE from Natural Language ✅	Week 6
7.	Final testing, debugging and reviewing the code for Demo	Test queries, optimize performance, add error handling ✅	Week 7-8
8.	In-Class Demo	Done perfectly ✅	Week 9
9.	Final report preparation and submission	Ongoing ✅	Week 10-11

Architecture Design



Flow Diagram

The architecture begins with user input being collected via a Streamlit-based web interface. The application extracts the schema of the connected MySQL database, which includes table names and their respective columns. This schema is bundled into the prompt sent to the Gemini LLM, which generates a corresponding SQL query. Once received, the query is cleaned to remove markdown artifacts and executed on the database. The resulting output is either a data table or status message and is then presented back to the user.

Implementation

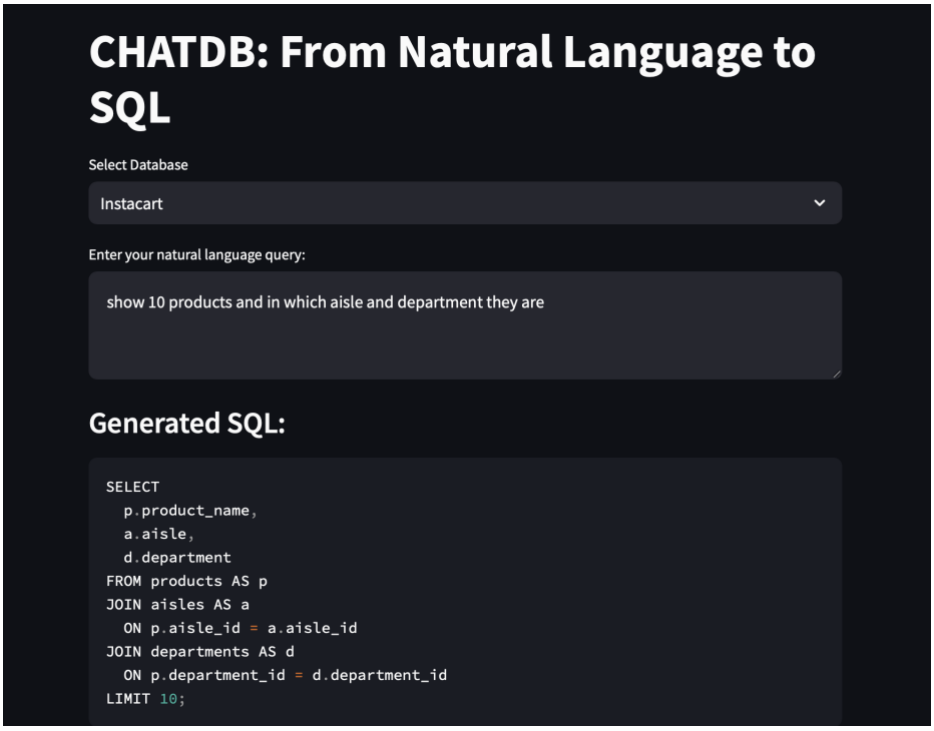
Functionalities

The application allows users to input queries in natural language and returns results fetched from a structured MySQL database. It automatically interprets user queries using Google Gemini and converts them into executable SQL. The system supports a wide range of SQL operations including SELECT, JOIN, WHERE, GROUP BY, and data modification commands like INSERT, UPDATE, and DELETE. Schema information is retrieved dynamically at runtime and appended to the prompt for Gemini, improving query accuracy. Error messages are returned for failed queries, and results are displayed in a user-friendly table format.

Tech Stack

This project was implemented entirely in Python. The frontend interface was created using Streamlit, which offers a simple and reactive environment for user interactions. Google Gemini API was accessed through the google-generativeai SDK to process natural language and generate SQL. For database operations, mysql-connector-python was used to communicate with a locally hosted MySQL database managed using MySQL Workbench. The schema is extracted programmatically to support context-aware prompt creation. Regex (re) was used to clean and sanitize Gemini outputs for reliable SQL execution.

Implementation Screenshots



```
FROM products AS p
JOIN aisles AS a
  ON p.aisle_id = a.aisle_id
JOIN departments AS d
  ON p.department_id = d.department_id
LIMIT 10;
```

Execute SQL Query

Query Results:

	product_name	aisle	department
0	"Chocolate Sandwich Cookies"	cookies cakes	snacks
1	"All-Seasons Salt"	spices seasonings	pantry
2	"Robust Golden Unsweetened Oolong Tea"	tea	beverages
3	"Green Chile Anytime Sauce"	marinades meat preparation	pantry
4	"Pure Coconut Water With Orange"	juice nectars	beverages
5	"Light Strawberry Blueberry Yogurt"	yogurt	dairy eggs
6	"Sparkling Orange Juice & Prickly Pear Beverage"	water seltzer sparkling water	beverages
7	"Peach Mango Juice"	refrigerated	beverages
8	"Fresh Scent Dishwasher Cleaner"	dish detergents	household
9	"Overnight Diapers Size 6"	diapers wipes	babies

CHATDB: From Natural Language to SQL

Select Database

library_management

Enter your natural language query:

Top 3 members who have borrowed the most number of books and also mention the number of books

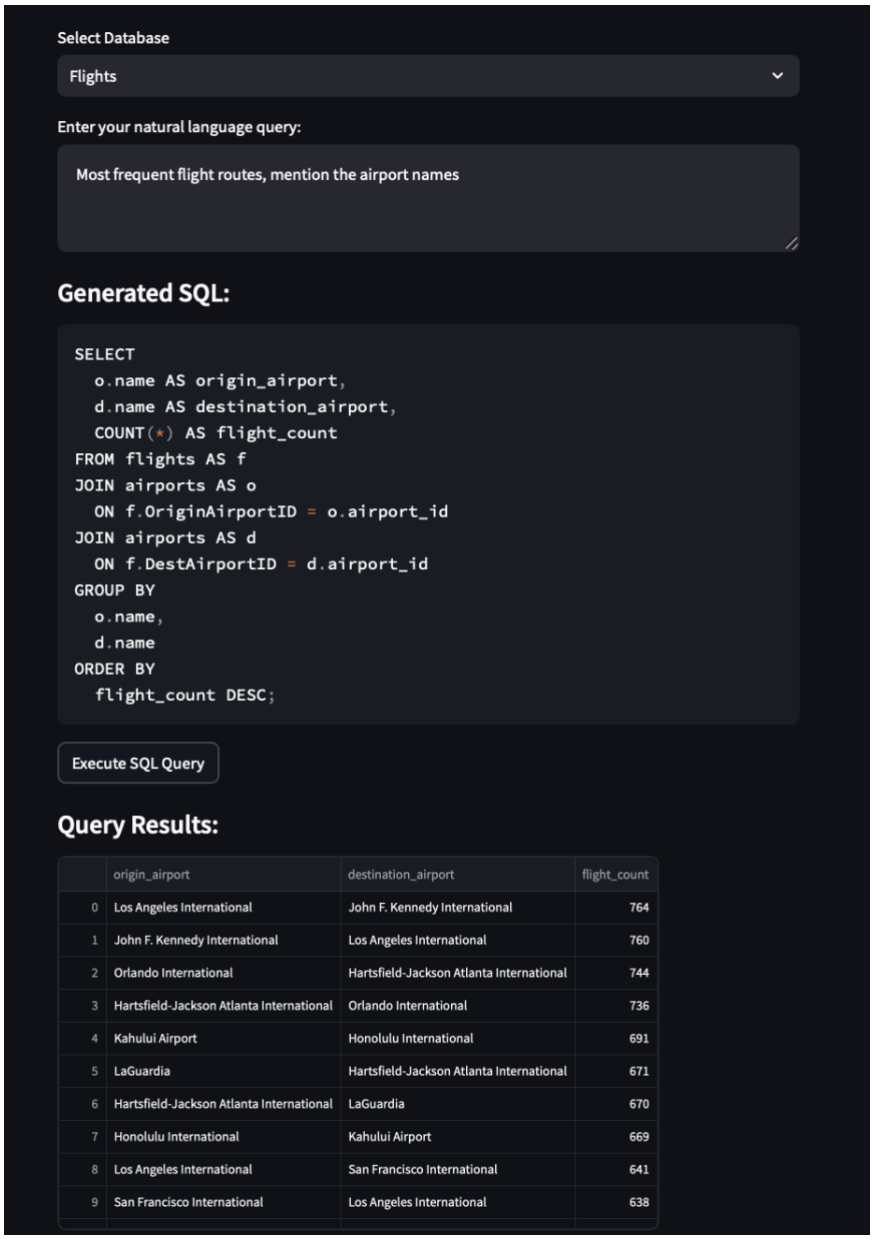
Generated SQL:

```
SELECT
  m.member_name,
  COUNT(l.book_id) AS num_borrowed_books
FROM Members AS m
JOIN Loans AS l
  ON m.member_id = l.member_id
GROUP BY
  m.member_name
ORDER BY
  num_borrowed_books DESC
LIMIT 3;
```

Execute SQL Query

Query Results:

	member_name	num_borrowed_books
0	Ashley Owen	7
1	Donald Torres	5
2	Rhonda Mills	5



Learning Outcomes

Through this project, I gained experience in working with large language models in a practical setting. I learned to engineer prompts that can guide LLMs toward more accurate outputs, especially in the context of SQL generation. I also became more familiar with Streamlit for rapid application development and improved my understanding of MySQL schema design and SQL debugging. Lastly, I deepened my knowledge in handling database connections securely and efficiently within a Python-based system.

Challenges faced

One major challenge was the Gemini model returning SQL functions not compatible with MySQL, such as STRFTIME() from SQLite. To fix this, I had to post-process the generated SQL or update the prompt to specifically request MySQL-compatible syntax. Another challenge was handling vague natural language queries that could be interpreted in multiple ways. This required careful schema design and detailed prompts. Lastly, cleaning and parsing Gemini's output—which sometimes included Markdown formatting like triple backticks—was necessary to avoid SQL execution errors.

Conclusion

This project successfully demonstrates the practical use of LLMs in making structured data accessible to non-technical users. By abstracting SQL through natural language, ChatDB significantly lowers the barrier to data querying. The system is modular, extendable, and has wide applications in business analytics, reporting, education, and data democratization platforms.

Future Scope

In the future, the system could be enhanced by integrating voice input for accessibility and speed. It could also support multiple databases such as PostgreSQL, Snowflake, and Oracle. Role-based authentication can be added to restrict sensitive queries.

Additionally, query logging and user analytics could be introduced to refine model prompts based on usage patterns. Data visualization tools like charts and graphs could also be incorporated for better interpretation of results.