ChatDB - A Python-Powered Conversational Interface for Learning SQL Queries

Mandy Chen (Working as solo)

Mandy holds a Bachelor's degree in Mathematics and Applied Economics from the University of California, Riverside, and is currently pursuing a Master's in Applied Data Science at the University of Southern California's Viterbi School of Engineering. With a strong foundation in analytical thinking and data management, she has developed proficiency in SQL database management. While her undergraduate studies involved minimal coding, she has since gained valuable programming skills, including Python for data manipulation, basic SQL query writing, and backend development using Flask.

Table Of Contents

Introduction

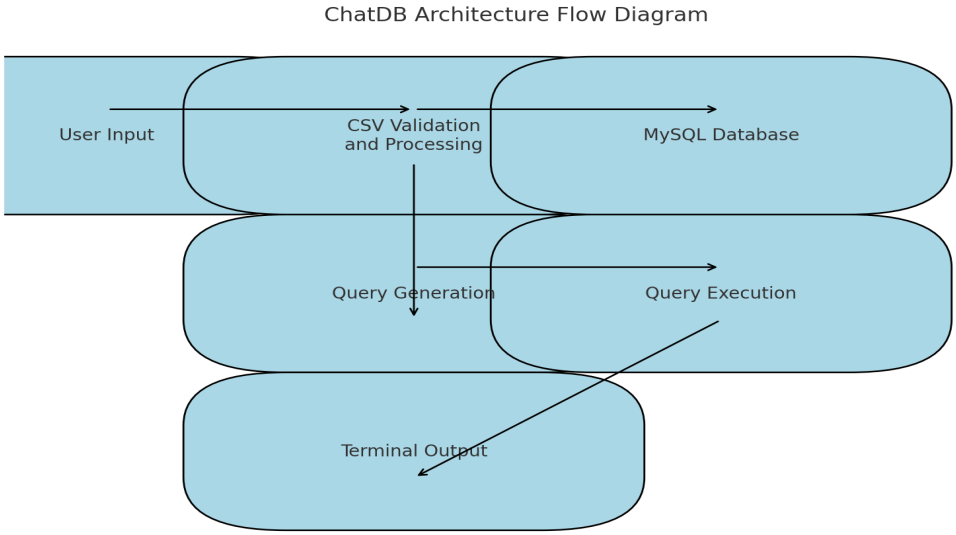
ChatDB is an interactive learning application designed to help users understand and practice SQL queries in an engaging way. By uploading .csv datasets, users can explore and execute SQL queries, with ChatDB providing query generation, execution, and explanations. As a solo project, ChatDB focuses on core functionality and supports five key SQL constructs: WHERE, GROUP BY, ORDER BY, HAVING and combined WHERE AND HAVING. For example, users may ask, “Find me sales where the store location is XYZ,” and ChatDB will dynamically generate the corresponding SQL query, explain its components, execute it on MySQL, and display the results.

Background and Motivation

SQL is a critical tool for data professionals, yet its syntax and structure often pose challenges for beginners. Many existing learning methods fail to connect theory with practical application, leaving learners struggling to grasp real-world use cases. This project was motivated by the need for a more engaging, hands-on way to learn SQL, particularly for those who prefer experimenting with their own data.

ChatDB addresses this need by enabling users to upload and explore .csv datasets in an interactive environment. Using datasets like the Chinook Music Store and Coffee Shop Sales, the application emphasizes immediate feedback, offering opportunities to practice SQL through queries generated and executed dynamically. This approach bridges the gap between traditional SQL learning tools and real-world data analysis, fostering confidence and deeper understanding.

System Design (Architecture Design — Flow Diagram and its Description)



ChatDB is designed as a lightweight, terminal-based application that provides users with a seamless way to practice and understand SQL queries. The system consists:

1. Data Processing Module:
    a. Users upload a .csv file, which is processed by Python to create a corresponding table in a MySQL database.
    b. Uploaded data is automatically inserted into the MySQL table, preparing it for exploration and querying.
2. Query Generation and Execution Module:
    a. The system supports five SQL constructs: WHERE, GROUP BY, ORDER BY, HAVING, and combined WHERE AND HAVING.
    b. Users may choose to explore their data using:
        i. Exploring datasets by viewing table and contained attributes
        ii. Pre-generated sample queries
        iii. Specific query constructs for hands-on learning
        iv. Natural language input, which is parsed into SQL queries
    c. SQL queries are executed on the MySQL database, and results are displayed directly in the terminal window.
3. Explanation Module:
    a. ChatDB provides explanations for each generated query, detailing its purpose to enhance user understanding.

Workflow Description

1. Dataset Uploaded: Users upload .csv files to the application.
2. Data Processing: The .csv file is processed, and a table is created in MySQL.

3. Query Interaction:
    a. Users can explore data through data exploration functions, pre-generated queries or specific SQL constructs.
    b. Users may input natural language.
4. Query Execution and Output:
    a. ChatDB generates the corresponding SQL query, executes it on the MySQL database, then displays results.
    b. ChatDB explains what the query does, reinforcing users learning experience

Test Datasets

ChatDB was tested with three datasets: Two of the Chinook Music Store (databases simulating a retail environment with customer, sales, and product data) and one Coffee Shop Sales (dataset with sales transactions, store locations, unit price etc., userful for exploring groupings, filters, and aggregations).

Design Considerations

1. Lightweight Design: The application runs entirely in the Python terminal using VS Code, ensuring simplicity without requiring additional frameworks.
2. Core Functionality Focus: Prioritized essential SQL constructs to maintain a balance between usability and feature complexity.
3. User-Friendly Interaction: Emphasized query explanations and practical uses.

Implementation

Overview

The system uses Python for its backend logic, integrating libraries like mysql.connector, pandas, and SQLAlchemy for database connectivity, data manipulation, and query execution. Users interact with the application through the terminal, where they upload .csv files and explore SQL concepts using pre-generated or dynamic queries.

Tech Stack (Libraries and Tools)

1. MySQL: Used as the database for storing and querying uploaded data.
2. Python

   a. Pandas for reading and processing .csv files.
   b. SQLAlchemy for database connectivity and table creation.
   c. mysql.connector and pymysql for executing SQL queries.
3. VSCode: Used as the development and runtime environment.

Key Functionalities

1. CSV Validation and Upload:
   a. Ensures the uploaded file is a valid .csv.
   b. Infers column types (INT, DECIMAL, VARCHAR, etc.) and generates a MySQL table schema.
   c. Handles edge cases, such as unsupported data types and empty columns.

```
Welcome to ChatDB!
Please upload your CSV file you would like to explore on: Coffee Shop Sales.csv
Table 'Coffee_shop_sales' already exists in the database. Skipping table creation.
Data inserted into table 'Coffee_shop_sales' successfully!
```

2. SQL Query Generation:
   a. Dynamically generates queries for supported constructs:
      i. WHERE: Filters rows based on conditions.
      ii. GROUP BY: Aggregates data by specific columns.
      iii. ORDER BY: Sort data in ascending order or descending order.
      iv. HAVING: Filters aggregated groups based on conditions.
      v. WHERE AND HAVING: Combines filterings of rows and the aggregated groups.
   b. Provides a natural language explanation for each query.

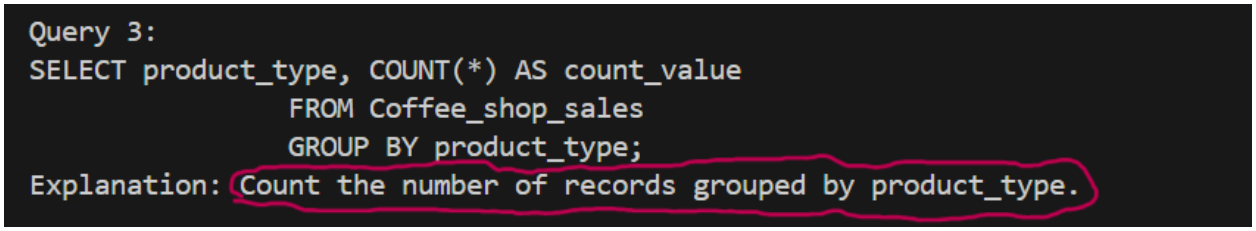
```
If you still like to explore more, please give me the number of the options or type 'exit' to leave ChatDB.

#1. Eplore your database. (See your table , attributes, data, etc.)

#2. Obtain sample queries.

#3. Obtain sample queries with specific language constructs.

#4. Ask specific queries regarding your data file.

Enter the number of your choice: 3

Which language constructs would you specifically like to explore?
1. WHERE
2. GROUP BY
3. ORDER BY
4. HAVING
5. WHERE AND HAVING
```

3. Query Execution:
   a. Executes the generated SQL query and fetches results.

b. Displays only the first five rows to ensure the output remains readable in the terminal.

```
Enter the number corresponding to your choice: 2

You selected the GROUP BY construct. Generating queries...

Generated Queries:
Query 1:
SELECT product_type, AVG(transaction_qty) AS avg_value
                FROM Coffee_shop_sales
                GROUP BY product_type;
Explanation: Calculate the average value of transaction_qty grouped by product_type.

Results (Noticing that I am only displaying you the preview of the first 5 rows):
('Gourmet brewed coffee', Decimal('1.5358'))
('Brewed Chai tea', Decimal('1.5277'))
('Hot chocolate', Decimal('1.5222'))
('Drip coffee', Decimal('1.5207'))
('Scone', Decimal('1.0287'))
--------------------------------------------------
Query 2:
SELECT product_detail, SUM(transaction_qty * unit_price) AS calculated_value
                FROM Coffee_shop_sales
                GROUP BY product_detail;
Explanation: Calculate the total value of transaction_qty * unit_price grouped by product_detail.

Results (Noticing that I am only displaying you the preview of the first 5 rows):
('Ethiopia Rg', Decimal('26358.00'))
('Spicy Eye Opener Chai Lg', Decimal('27304.80'))
('Dark chocolate Lg', Decimal('42012.00'))
('Our Old Time Diner Blend Sm', Decimal('17936.00'))
('Oatmeal Scone', Decimal('10920.00'))
--------------------------------------------------
Query 3:
SELECT product_type, COUNT(*) AS count_value
                FROM Coffee_shop_sales
                GROUP BY product_type;
Explanation: Count the number of records grouped by product_type.

Results (Noticing that I am only displaying you the preview of the first 5 rows):
('Gourmet brewed coffee', 33824)
('Brewed Chai tea', 34366)
('Hot chocolate', 22936)
('Drip coffee', 16954)
('Scone', 20346)
--------------------------------------------------
```

4. Natural Language Processing/Handling:
    a. Users may ask queries in natural language (e.g., “Find sales where store location is XYZ”).
    b. ChatDB parses the input, maps it to the SQL query and executes it.



Learning Outcomes (Challenges and Solutions)

One of the biggest challenges I faced while building ChatDB was handling natural language processing to dynamically generate queries for different datasets. Since users can upload .csv files with unique structures and column names, hardcoding specific queries wasn't practical. For example, if a user uploads a file with columns like "price" and "color" and wants to find the price of blue items, the query should be SELECT price GROUP BY color. To make this work, I had to detect patterns in the user's input, identifying the word before GROUP BY (like "price") as variable A and the word after it (like "blue") as variable B. This way, whenever GROUP BY was triggered, the system could generate a valid query based on the table structure. To make this process smoother, I designed query templates and gave users examples to follow, which helped the system understand their intent more easily. Developing this pattern detection system was challenging, but it taught me a lot about making applications more flexible and user-friendly.

Testing and Validation

ChatDB was tested with three datasets described under test datasets of system design. Test cases covered uploading valid and invalid .csv files, generating and executing queries for all supported constructs, handling edge cases, such as empty tables or ambiguous column names.

Limitations

1. The system currently supports only .csv files and MySQL as the database
2. Complex queries, such as nested or multi-join queries, are not supported.
3. Natural language input relies on predefined templates, limiting its flexibility

Potential Future Enhancements

1. Expand natural language processing to handle more diverse query structures.
2. Add support for additional data formats such as .json(MongoDB), .xlsx.
3. Integrate a lightweight web interface for improved usability

Conclusion

ChatDB successfully demonstrates an interactive approach to learning SQL by allowing users to upload .csv files, automatically process them into a MySQL database, and explore data through dynamically generated or user-specified queries. By supporting different kinds of SQL constructs and providing natural language query handling, ChatDB bridges the gap between theoretical learning and practical application. While ChatDB is currently limited to .csv files and a

terminal-based interface, the project serves as a strong foundation for future enhancements, such as expanded query capabilities and a more user-friendly interface.

Future Scope

In the future, I want to expand ChatDB to support more SQL language constructs beyond the five it currently handles. I'd like to include more advanced features like joins, subqueries, and window functions to make the system even more versatile for users. Adding these constructs will require a strong foundation, especially since every new query relies on natural language processing and accurate pattern detection. Working on the current constructs has taught me how important it is to build a flexible and scalable system. To move forward, I plan to improve the query generation process, enhance the NLP capabilities to handle more complex inputs, and design clearer templates to help users express their queries effectively. This will allow me to make ChatDB a more powerful and comprehensive tool for learning SQL.