ChatDB39 Project Midterm Report

Project Overview

This project aims to create a ChatDB, a database interface that allows users to interact with multiple databases using natural language queries. The system must support three different database instances (e.g., **MySQL, FireBase, and MongoDB,** specifically for our project) and correctly handle queries, joins, and modifications across all databases.

When complete, the system will include:

- **A natural language interface (NLI)** that translates user queries into executable SQL or NoSQL queries.
- **Support for three database instances**: MySQL (SQL), Firebase (NoSQL), and MongoDB (NoSQL).
- **A web-based UI** to facilitate user interaction.
- **A unified backend API** that can use the result from NLI to support multiple database instances and handle queries, joins, and modifications across all databases.

Team Members

Member 1: Yijian Jin

- Background: First-year graduate student majoring in Applied Data Science; with a bachelor's degree in Statistics and Data Science. Previous relevant experience includes statistical modeling, ML, and Python/R data processing.

Member 2: Hongyu Yu

- Background: First-year graduate student majoring in Applied Data Science. My undergraduate major was applied mathematics, and I also had a minor in data science. Previous experiences included MySQL, Python, machine learning, and some Java.

Member 3: Zilu Wang

- Background: Second-year graduate student majoring in Computer Science. Previous experience includes Machine Learning, Computer Vision, and software engineering.

Implemented Questions

Tech Stack Used

- Frontend**:** React.js for user interface and UI
- Backend: FastAPI, Google Gemini API
- Database Connectivity: PyMySQL, PyMongo, Firebase Admin
- Other Essential Libraries: Pandas, Regex (re), SQLAlchemy, uvicorn, re, JSON, etc.

Query Syntax Implementation Plan

- The frontend receives NL queries from the user
- The backend routes NL queries to the Google Gemini 2.0 Flash model, generating SQL/NoSQL queries.
    - Query conversion follows predefined rules to route specific queries to MySQL, MongoDB, or Firebase.
- A query parser validates syntax and routes queries to the desired databases.
- The backend executes the query and combines results from databases. It then returns JSON for frontend display.

- The frontend needs to handle the input and display the output for the whole process.

Database Selection

- **MySQL**: This handles structured relational data.
- **MongoDB**: Stores document-based (Binary JSON) NoSQL data.
- **Firebase Realtime Database**: JSON-based NoSQL for real-time synchronization data storage.

Planned Implementations

1. What have we done?
    a. Natural language query processing
        i. Use Google Gemini API to convert NL queries to SQL/NoSQL queries.
        ii. Query types can be identified most often and routed to the appropriate database.
    b. Backend
        i. Handle user requests, process queries, and return structured responses.
        ii. integrates with database connectors for MySQL, MongoDB, and Firebase.
    c. Database Implementation
        i. sample data collection: Airbnb listings in LA (75 columns, over 40000 rows)
            1. Data cleaning (some listings that miss important data are removed; a small portion of random samples is selected (500 rows) for backend testing purposes)
        ii. MySQL hosts structured data like listings, features, and hosts.
        iii. MongoDB stores unstructured data such as listing descriptions, reviews, listing links, and amenities in document format.
        iv. Firebase manages real-time availability and pricing.
    d. Query Execution
        i. Translates natural language queries into SQL for MySQL and JSON-based filters for MongoDB and Firebase.
        ii. Executes queries on multiple databases and merges results where applicable.
    e. Testing
        i. A basic test script is implemented to verify query processing and execution
    f. FrontEnd
        i. Complete frontend framework setup and UI design (including search input zone and result displaying zone)
        ii. Complete reserved interfaces
        iii. Complete frontend mock data testing

2. What is different from the original proposal?
    a. Switched from ChatGPT to Google Gemini for cost-effectiveness
    b. Switched from query templates to full LLM-based query translation
    c. Improved error handling with try-except blocks and implemented query caching for more intuitive performance tracking

Project Status

1. What is already implemented?
    a. Database setup

        i. python scripts that load data to local databases (MySQL, MongoDB) and Firebase.
        ii. basic data cleaning to handle missing values.
    b. Query Conversion
        i. Google Gemini API is successfully integrated.
        ii. Query routing logic is implemented to identify whether a query should go to MySQL, MongoDB, or Firebase.
    c. Backend
        i. The FastAPI backend is implemented.
        ii. Database connectors integrated for MySQL, MongoDB, and Firebase.
        iii. Send, receive, and execute queries on relevant databases and return results.
    d. Testing
        i. A basic test script is implemented, which can run sample queries and output results.
        ii. Basic error handling is implemented in case of failed queries or invalid responses from Google Gemini.
2. What will we do next?
    a. Improve query validation to ensure queries generated by LLM are always executable.
    b. Implement pre-execution query to check response correctness and avoid unnecessary database calls.
    c. Improve the way MySQL, MongoDB, and Firebase results are combined.
    d. Run extensive tests on query accuracy and failure handling. Test edge cases where multiple query requests are tested at once.
    e. Improve the FrontEnd, Frontend, and backend integration, then do the unit and joint tests.
    f. Enhance frontend and Backend security.

Challenge Faced

1. NLP to SQL/NoSQL Conversion Issues

- **Problem:** The AI model may not be "smart," user queries may be ambiguous, and complex queries might generate incorrect SQL/NoSQL queries. For example, generated queries for MongoDB and Firebase sometimes use incorrect syntax or fail to match actual database structures.
- **Solution:**
    - Guide the user to input more accurate natural language input and give more context, such as returning a default query template for the user to rephrase their request;
    - Refine the LLM prompt with examples and give more structured instructions to the LLM;
    - Automate query correction where the program can send the problematic query back to LLM with additional instructions, creating a self-correcting loop to improve reliability without user intervention.
    - implement a syntax checker before execution.
    - Switch to paid models;

2. Frontend-Backend Integration Issues

- **Problem:** Slow query execution, authentication failures, or frontend request format doesn't match the backend interface.

- **Solution:** Use accurate and standard API documents and use Postman to test every interface. The frontend should mock data to display for the test.

3. Database Consistency Challenges

- **Problem:** Syntax differences between SQL and NoSQL databases, JOIN issues in NoSQL, data format, and content inconsistency across different databases.
- **Solution:** Design and improve a query router class to handle SQL/NoSQL dynamically; ensure shared fields (e.g., id, price) have consistent types and naming conventions across databases; Improve ID mapping logic to ensure that MySQL and MongoDB results can be correctly merged.

4. Data Processing problems

- **Problem:** Inconsistent field formats across databases, such as timestamps or data lists;
- **Solution:** Standardize field formats, and design good helper functions and standards for data transformation; Use pandas to preprocess inconsistencies before loading data to databases;

5. Security issues

- **Problem:** SQL injection, unauthorized access, or other network or database attacks.
- **Solution:** Use parameterized queries to avoid SQL injection and JWT authentication (if necessary) and enforce interface implementations' security and code styles.

Timeline

Goals	**Tasks**	**Completion Time**	**Status**
Proposal Submission	Formulate and distribute detailed tasks for every group member.	Feb 7	Complete
Initial Setup	Setup Database & API framework	Feb 20	Complete
NLP Query Processing	Implement basic query conversion	Mar 5	Complete
Midterm Progress Report	Progress report & troubleshooting	Mar 7	Complete
Advanced Features	Fine-tune NLP, optimize system	Apr 1	In progress
UI Development	Frontend & user interaction	Apr 10	In progress
Testing & Debugging	Refine and fix issues	Apr 15	
Final Demo	Present & showcase system	Apr 21 and 23	
Final Report	Submit final documentation	May 9	