DSCI 551 - Project Guideline

Spring 2025

ChatDB – Taking to Database Management Systems using Natural Language

In this project, you are asked to develop a natural language interface to one or more SQL/NoSQL database systems (depending on the number of people in your group). The interface should support the following functions for RDBMS (e.g., MySQL):

- Explore schemas and data of database: It should allow users to ask questions (in natural language) to find out what tables there are in the database, what attributes a table has, and get sample data from a table.
- Query: it should accept natural language queries from users, and convert them into database queries (SQL), execute the queries in the DBMS, and return query results to the users. It should support the common constructs in SQL, including select, from, where, group by, having, order by, limit, and offset. It should also allow queries that involve joining multiple tables.
- Data modification: It should support insert, delete, and update requests in natural language. For example, add a new employee named John in the HR department; or update John's age to 26.

Similarly, the interface should support the following functions for NoSQL databases (e.g., MongoDB):

- Explore schemas and data of database: It should allow users to ask questions to find out what *collections* there are in the database and get sample data from a table.
- Query: it should accept natural language queries from users, and convert them into database queries, execute the queries in the DBMS, and return query results to the users. It should support the functions in MongoDB: find (with projection), aggregate (with $match, $group, $sort, $limit, $skip, $project). Note that $match can be before and after $group. It should also allow queries that involve joining of two collections (using $lookup).
- Data modification: It should support insert, delete, and update requests in natural language. For example, add a new employee named John in the HR department; or update John's age to 26. ChatDB should convert the requests into MongoDB functions such as insertOne, updateOne, deleteOne, insertMany, etc.

Additional Requirements and Notes:

- You can form a group of up to 3 people for your project.
    - One-person group: You can develop a natural language interface (NLI) to either a RDBMS (except for sqlite) or a NoSQL database.
    - Two-person group: You should develop NLI to a RDBMS and a NoSQL database.
    - Three-person group: You should develop NLI to a RDBMS and a NoSQL database. In addition, you should develop a Web browser-based UI for users to interact with ChatDB.

- You can use OpenAI or a large language model API for the projects. See the appendix for possible options and some notes which you might find helpful in your project implementation. You are encouraged to share your experiences and further resources on LLM API on Piazza!
- Use at least three databases to showcase the working of your ChatDB implementation. Note that you will need to have at least two tables/collections to demonstrate join.

Project deliverables:

- Proposal (due 2/7, Friday, 10 points): Detail your project plan, including type of database system, databases, and how you plan to implement the functions stated in the beginning of this handout. Also list group members and their roles. Note that your project is a collaborative effort, and all members should contribute to the project. We will grade your project as a team effort and every member receives the same grade.
- Midterm progress report (due 3/7, Friday, 10 points): tell us your progress so far and the challenges you might have encountered. Note that any reformation of groups should be made before midterm progress and reported in the midterm progress report (also notify TA). It will be the responsibility of all group members to make sure the project will be a team effort after that.
- Demo (in-class, 4/21 and 4/23, 10 points): Give a live demo of your project. All project members should be present during the demo, presenting his/her contribution. If you are absent, we will assume that you have not contributed to the implementation of the project.
- Final report (due on 5/9, Friday, 10 points): the final report should be comprehensive, details your design and implementation, and your learning experiences.
- Implementation (due on your demo time, 60 points): note your project should be fully implemented before the demo. You should include in your final report a link to Google drive where you will upload your project codebase and documentations. Make sure you give access to your project folder.

Appendix: Some Options for LLM API (be sure to check out respective websites for more accurate and up-to-date details and pricing)

- **Free/Open-Source Options**
  - **Local LLMs**

    1. **Llama 2 (Meta)**
       - Versions: 7B, 13B, 70B parameters
       - Hardware Requirements: 8GB–48GB RAM
       - Meta AI Llama
       - Python Implementation: **llama-cpp-python**

    2. **Mistral 7B**
       - Optimized for efficiency
       - Minimum Hardware Requirements: 8GB RAM
       - Mistral AI
       - Easy deployment via HuggingFace

- **Free APIs**
  1. **Hugging Face Inference API**
     - Free Tier: 30K requests/month
     - Access to multiple pre-trained models
     - Hugging Face API

- **Paid Options**

  OpenAI
  1. **GPT-3.5 Turbo**
     - It appears to have free Credits: $5 for first three months (new users)
     - OpenAI Pricing
     - Rate Limits: 3 requests/minute on free tier

  2. Also check out the mini version of the latest GPT-4o, which might have lower costs ($.15 per 1M input tokens, $.6 per 1M output tokens).

  Anthropic Claude
  1. **Claude API**
     - Competitive pricing with academic discounts available
     - Python SDK support
     - Anthropic Pricing

  Azure OpenAI
  1. **Azure Cognitive Services**
     - Benefits: Institutional pricing and free credits for students
     - Integration with Azure services
     - Azure OpenAI Service
     - Requires application approval for access

- **Implementation Recommendations**
  - Set query limits to manage costs and API calls.
  - Use caching mechanisms to avoid redundant queries.
  - Implement batch processing for bulk operations.
  - Monitor token usage to ensure efficiency.