Minor Project Report

on

DraftAi, Incubated at Atal Incubation Center, GGSIPU

Submitted in partial fulfillment of the requirements for the completion of Minor Project [ARP 455]

Name: Pratham Kumar

Enrollment Number: 00119011921

Under the supervision of

Dr. Manisha Parlewar

UNIVERSITY SCHOOL OF AUTOMATION AND ROBOTICS
GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY
EAST DELHI CAMPUS, SURAJMAL VIHAR, DELHI- 110032

Table of Contents

List of Figures

Abbreviations and Nomenclature

Abbreviation	Definition
SQL	Structured Query Language
REST	Representational State Transfer
LLM	Large Language Model
BERT	Bidirectional Encoder Representations from Transformers
API	Application Programming Interface
GPT	Generative Pre-trained Transformer
CSV	Comma Separated Values
NLP	Natural Language Processing
IDE	Integrated Development Environment
SSD	Solid State Drive
RAM	Random-access memory
ASGI	Asynchronous Server Gateway Interface
HTTP	Hypertext Transfer Protocol
AI	Artificial intelligence

Table: 1 Abbreviation and Nomenclature Table

Abstract

QueryMate is an innovative application designed to bridge the gap between natural language and structured SQL query processing, enabling non-technical users to interact seamlessly with databases. By leveraging FastAPI for backend operations and Streamlit for frontend visualization, QueryMate converts natural language queries into executable SQL statements and delivers results in SQL and CSV formats. The platform integrates with SQLite databases and utilizes CSV files to support diverse data sources. Key features include memory persistence, intuitive error handling, and a robust REST API for query submission, which facilitate smooth and reliable interactions. Deployed on Hugging Face Spaces, QueryMate is accessible to users online, with Docker support for streamlined setup across environments. QueryMate's design addresses the challenges of real-time query generation, asynchronous API communication, and persistent chat history, making it a versatile tool for data access and exploration.

Introduction

In today's data-driven landscape, accessing and analyzing structured data is crucial for informed decision-making across industries. However, translating complex database queries into insights often requires specialized knowledge in SQL, limiting access for non-technical users. QueryMate aims to democratize data access by transforming natural language inputs into SQL queries, allowing users without SQL expertise to interact seamlessly with databases and obtain data-driven insights.

QueryMate leverages FastAPI for backend processing, which translates user queries into SQL and CSV outputs, and Streamlit for an intuitive frontend, enabling users to input queries, view results, and maintain chat history. This dual architecture ensures a robust and user-friendly experience, while support for SQLite and CSV files enables compatibility with various data sources.

By enabling users to ask questions in plain language and receive real-time data insights, QueryMate removes technical barriers, fostering a more accessible approach to data exploration and analysis. The application is deployed on Hugging Face Spaces, making it readily available, and Docker support further facilitates easy deployment across different environments. QueryMate thus represents a step forward in making data analysis more inclusive, empowering users to leverage data effectively for better, data-informed decision-making.

Hardware and Software Requirements

Hardware Requirements

1. **Processor:** Quad-core CPU or higher for faster response times when handling large datasets and complex queries, such as an Intel i5 12th generation or higher.
2. **Memory (RAM):** 8 GB or higher for smooth performance and improving multitasking capabilities.
3. **Storage:** SSD storage for faster read/write speeds, beneficial for querying large CSV files or databases.

Software Requirements

1. **Operating System:** Windows 10 or higher or Linux Suitable choices for most modern software development environments.
2. **Python**: Python 3.12 or higher is required for running FastAPI (backend), Streamlit (frontend), and handling machine learning and data processing tasks, ensuring compatibility with modern libraries and performance improvements.
3. **Python Libraries and Dependencies:**
    a. FastAPI: For building the REST API backend.
    b. Streamlit: For creating the web interface frontend.
    c. Uvicorn: ASGI server for serving the FastAPI application.
    d. Pandas: For CSV file processing and handling.
    e. SQLite3: Integrated within Python, used for handling SQL databases.
    f. google-generativeai: API client library for Google's generative AI models.
    g. Requests: For managing HTTP requests between frontend and backend.
    h. python-dotenv: For managing environment variables like API keys.
4. **External APIs:** Google Generative AI API Used to convert natural language queries into SQL or pandas queries. Requires API key setup in an environment variable.
5. **Deployment**: Docker, Enables containerization of the application for consistent performance across environments. Docker simplifies deployment with pre-configured environments and dependency handling.

Problem Statement

In the modern data-driven world, querying databases and analyzing data from CSV files is essential for making informed decisions. However, interacting with data often requires specialized knowledge of query languages such as SQL or pandas, which can pose a barrier for non-technical users. Many business users, analysts, and decision-makers may struggle with complex queries, limiting their ability to extract valuable insights from structured data.

The problem lies in the complexity of converting natural language questions into actionable data queries, especially for individuals without a background in programming or data analysis. While tools like SQL and pandas provide powerful capabilities for data manipulation, they often require users to have a deep understanding of syntax, query structure, and data relationships.

QueryMate aims to solve this problem by offering a solution that translates natural language queries into executable SQL or pandas code. The tool eliminates the need for users to learn complex syntax or query structures by allowing them to interact with data in a way that is both intuitive and accessible. By leveraging generative AI for query translation, QueryMate enables users to ask questions in plain English and receive results from a database or CSV file without writing code, democratizing access to data analysis and improving decision-making capabilities across various domains.

Literature Survey

The ability to convert natural language queries into structured queries, such as SQL (Structured Query Language), is a significant area of research in Natural Language Processing (NLP) and database management. Text-to-SQL systems aim to bridge the gap between human language and databases, allowing users to interact with data using natural language. Similarly, querying CSV files (which are commonly used for storing data in a tabular format) using natural language is also gaining attention, especially in the context of business intelligence and data analysis tools.

Text-to-SQL Systems

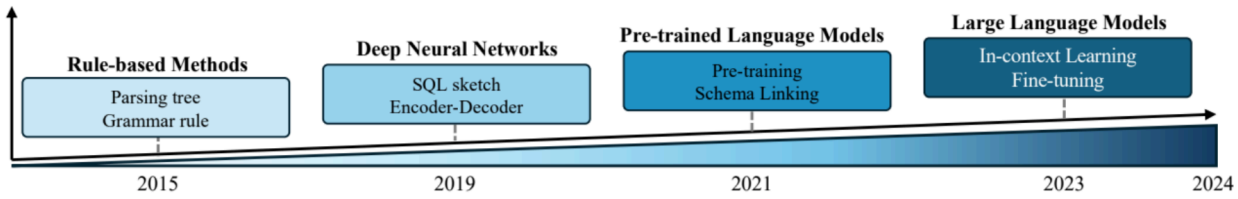

Fig1 : Evolutionary process for text-to-SQL research

1. Early / Rule-based Approach

Early attempts at text-to-SQL translation relied heavily on rule-based or template-based systems. These approaches mapped certain keywords or syntactic structures in natural language to predefined SQL templates. However, these methods were limited in their flexibility and scalability, as they required extensive manual effort to create rules and templates.

- **Advantages**: Easy to implement for simple queries with predefined templates.
- **Limitations**: Poor handling of complex queries, ambiguous natural language, and variation in sentence structures.

2. Statistical and Machine Learning-Based Methods

With the advent of machine learning, the focus shifted to more adaptive systems capable of learning patterns from large datasets. Early approaches like Seq2Seq models

(sequence-to-sequence learning) used recurrent neural networks (RNNs) to convert a natural language sentence into SQL queries. The key advantage of these systems was their ability to generalize over a wide range of sentence structures.

- **Advantages**: Improved accuracy and adaptability, can handle complex query structures.
- **Challenges**: Requires large amounts of training data, and can struggle with ambiguous queries.

3. Pre-trained Language Models (PLMs)

Recent advancements in NLP, particularly with transformer models like BERT, GPT, have brought significant improvements to text-to-SQL systems. These models, especially when fine-tuned on specific text-to-SQL datasets, have demonstrated remarkable performance.

- **Advantages:** High performance on large-scale datasets, can handle more varied and complex queries, ability to capture semantic nuances of natural language.
- **Challenges:** Still requires significant computational resources for training and inference, struggles with unseen databases or schemas.

4. Large Language Models (LLMs)

Recent research has focused on improving the ability of text-to-SQL systems to work with limited training data (few-shot learning) or even without explicit training on a specific task (zero-shot learning). LLMs like ChatGPT, Gemini have demonstrated few-shot capabilities, where the model can understand and generate SQL queries even for tasks it has not been explicitly trained on.

- **Advantages**: Better generalization to new databases and queries with minimal retraining.
- **Challenges**: May produce syntactically correct SQL queries that are semantically incorrect, especially when dealing with complex schema or rare entities.

CSV Query Systems

1. Traditional CSV Querying

Historically, querying CSV files involved using scripts or querying tools (e.g., Pandas in Python) to manipulate and analyze data. These systems required users to have programming or query language skills, which could be a barrier to non-technical users.

- **Limitations**: Requires users to have knowledge of programming languages or scripting tools, not intuitive for non-expert users.

2. Natural Language Querying of CSV Files

With the rise of NLP technologies, several systems have been proposed to enable natural language querying of CSV files. These systems aim to allow users to ask questions in natural language, which are then converted into operations (e.g., filtering, aggregation) on the CSV data.

- **Advantages**: Provides an intuitive interface for non-technical users to interact with CSV data without needing to learn SQL or programming languages.
- **Challenges**: Handling large datasets, ensuring efficient query execution, and maintaining the accuracy of natural language interpretation.

Major Modules of Project

- **Backend (FastAPI Module)**:

  It is responsible for processing user queries and interacting with either the database or CSV files. It handles the following responsibilities:

  1. Accepts queries from the frontend (Streamlit).
  2. Uses Google Generative AI to convert natural language queries into SQL or pandas queries.
  3. Executes generated SQL queries on the SQLite database.
  4. Executes generated pandas queries on the CSV file.
  5. Returns the query results to the frontend.

- **Generative AI Integration Module (Google's Gemini AI):**

  It utilizes Google's generative AI to transform natural language queries into executable SQL or pandas code. It handles the following responsibilities:

  1. Takes a user's input (natural language question) and converts it into SQL or pandas syntax based on the selected data source (SQL or CSV).
  2. Uses a predefined prompt for SQL queries and CSV queries to guide the AI model in generating accurate code.

- **Database Interaction Module (SQLite):**

  It Manages communication between the application and the SQLite database. It handles the following responsibilities:

  1. Executes SQL queries on the database.
  2. Handles database connection, query execution, and result fetching.
  3. Returns the results of SQL queries to the frontend.

- **CSV Interaction Module (Pandas):**

  It Manages interaction with the CSV file. It handles the following responsibilities:

  1. Executes pandas queries on the file based on AI-generated queries.

2. Returns results from the CSV file in a structured format (e.g., list of dictionaries, DataFrame).
3. Provides insights from the CSV file based on user queries.

- **Frontend (Streamlit Module):**

  It Provides an interactive user interface for users to submit queries and view results. It handles the following responsibilities:

  1. Displays the user interface for query input and results.
  2. Allows users to select between different data sources (SQL Database or Employee CSV).
  3. Sends user queries to the FastAPI backend for processing.
  4. Displays the generated SQL or pandas query and the query results (either in tabular format or as a list).

- **Deployment Module (Docker):**

  It ensures that the application is packaged and deployed correctly, whether on cloud platforms (like Hugging Face Spaces) or local Docker environments. It handles the following responsibilities:

  1. Ensures the application is packaged into Docker containers for deployment.
  2. Manages environment variables (such as API keys) and configuration settings.

- **Error Handling and Logging Module:**

  It ensures that errors are handled gracefully and important events are logged for debugging and monitoring. It handles the following responsibilities:

  1. Provides detailed error messages for issues encountered during query execution (SQL or pandas errors).
  2. Logs errors and key events for performance monitoring and debugging.

Snapshot of Web Portal / IDE

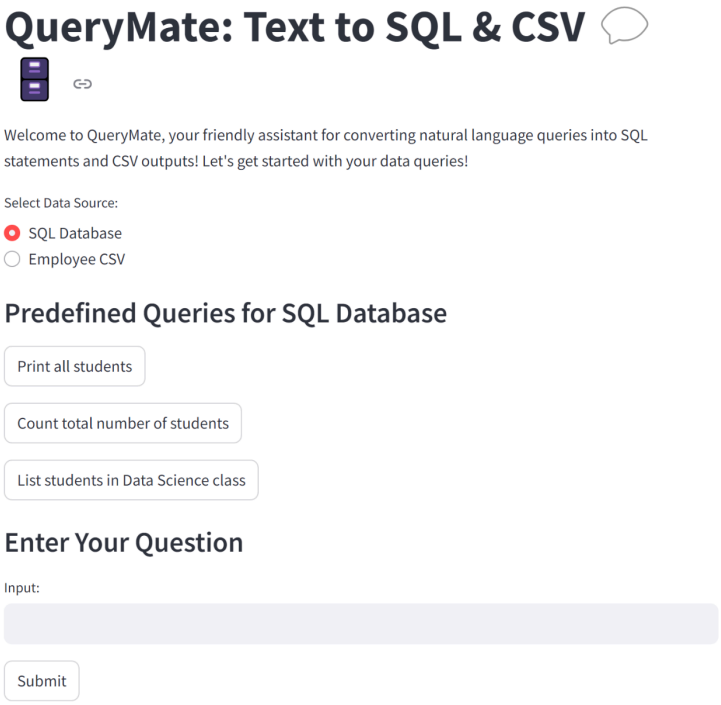

Fig 2: QueryMate: Text to SQL & CSV Home Page

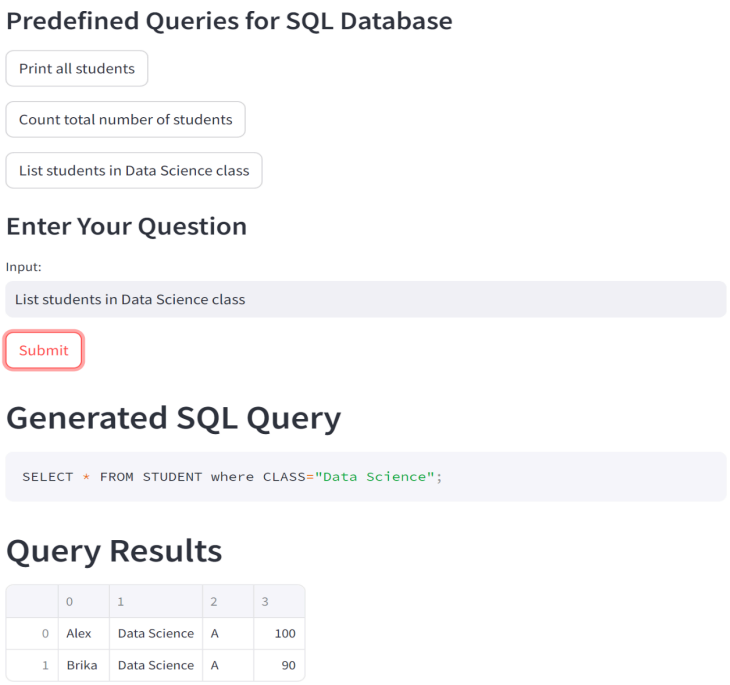

Fig 5: Choosing to SQL Database

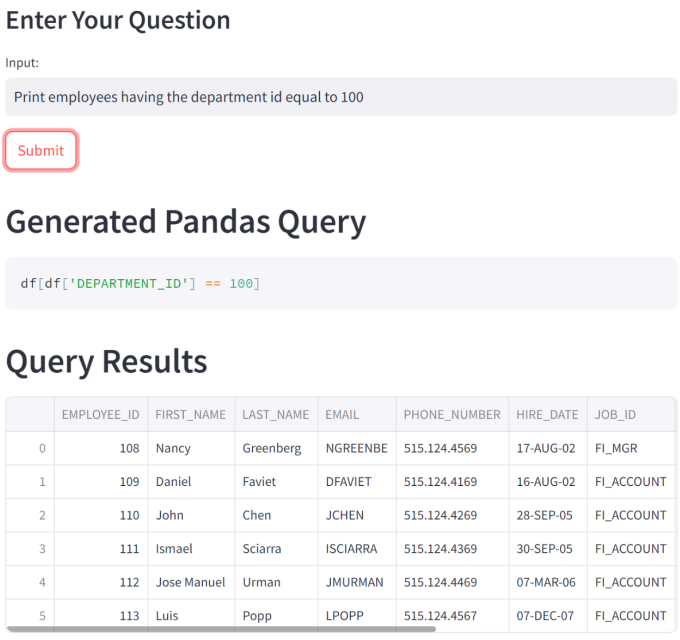

Fig 6: Choosing to CSV Database

Available CSV Columns


```
[
  0 : "EMPLOYEE_ID"
  1 : "FIRST_NAME"
  2 : "LAST_NAME"
  3 : "EMAIL"
  4 : "PHONE_NUMBER"
  5 : "HIRE_DATE"
  6 : "JOB_ID"
  7 : "SALARY"
  8 : "COMMISSION_PCT"
  9 : "MANAGER_ID"
  10 : "DEPARTMENT_ID"
]
```


Chat History

(SQL Database): List students in Data Science class

: SELECT * FROM STUDENT where CLASS="Data Science";

(Employee CSV): List Top 5 employees according to salary in descending order

: df.nlargest(5, ['SALARY'])

(Employee CSV): Print employees having the department id equal to 100

: df[df['DEPARTMENT_ID'] == 100]

Clear Chat History

Fig 7: Chat history

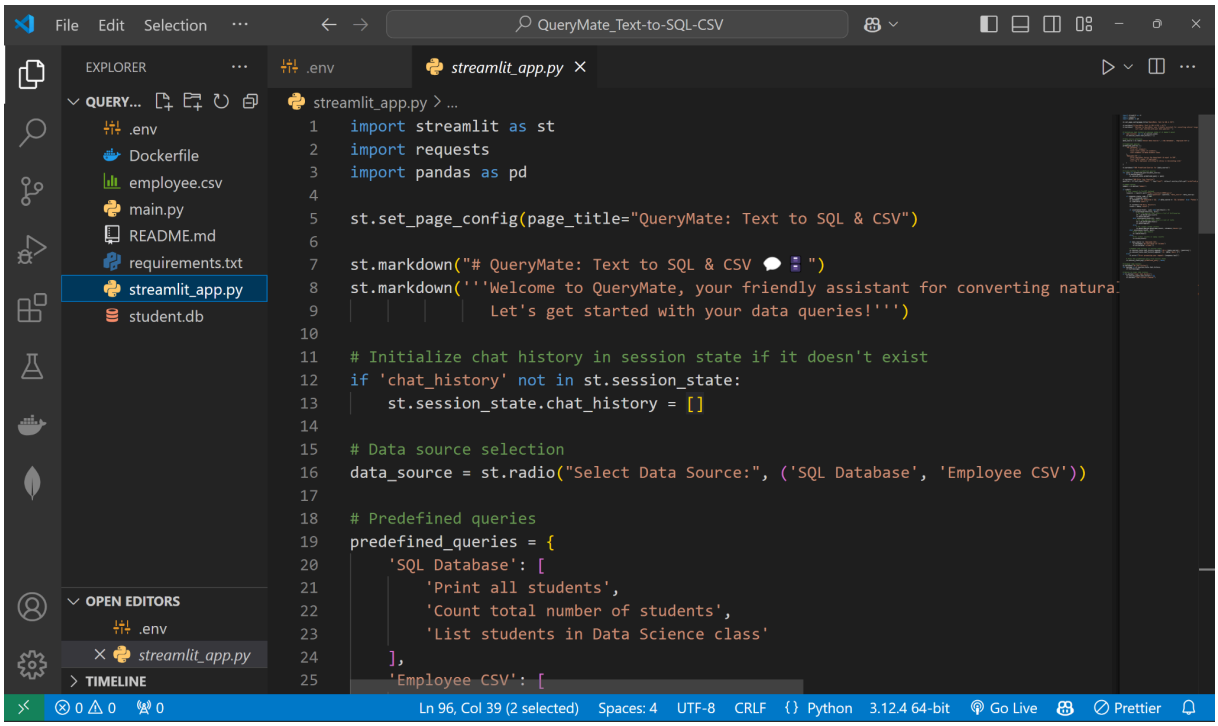

Fig 8: IDE Screenshot

Result Obtained

1. **Accurate Query Generation:** QueryMate effectively converted natural language queries into valid SQL or pandas syntax using Google Generative AI, achieving high accuracy in query generation across various test cases.

2. **Successful Query Execution:**

- **SQL Database:** Executed SQL queries on the SQLite database accurately, including counting entries, filtering by class, and retrieving specific student records.
- **CSV Data:** Executed pandas queries on the CSV file successfully, handling tasks like filtering employees by department, searching by employee ID, and sorting based on salary.

3. **User-Friendly Interface:** The Streamlit front-end provided an intuitive and accessible interface, displaying both the generated query and the query results clearly and allowing users to easily interact with the system.

4. **Effective Error Handling:** Robust error handling mechanisms delivered clear feedback for invalid queries, enhancing reliability and helping users understand input errors.

5. **Session Context:** Maintained a chat history for each session, improving user experience by allowing users to track and revisit previous queries and results.

6. **Scalable Deployment:** Successfully deployed on Hugging Face Spaces, ensuring that QueryMate is accessible and performs consistently across various environments, with additional Docker support for local or cloud deployment.

References/Bibliography

1. Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL
   Zijin Hong1, Zheng Yuan2, Qinggang Zhang2, Hao Chen2, Junnan Dong2, Feiran Huang1, and Xiao Huang21
   https://arxiv.org/html/2406.08426v1

2. Xu, X., Liu, C., & Song, D. (2017). SQLNet: Generating Structured Queries from Natural Language Without Reinforcement Learning. *arXiv preprint arXiv:1711.04436*.
   https://arxiv.org/abs/1711.04436

3. Hugging Face documentation: https://huggingface.co/docs
4. Gemini API documentation: https://ai.google.dev/gemini-api/docs
5. FastAPI documentation: https://fastapi.tiangolo.com/tutorial/
6. Docker documentation: https://docs.docker.com/
7. SQLite documentation: https://www.sqlite.org/docs.html
8. Streamlit documentation: https://docs.streamlit.io/
9. Pandas Tutorial: https://www.w3schools.com/python/pandas/default.asp
10. docker tutorial for ML: https://youtu.be/h5wLuVDr0oc?feature=shared
11. Request Python Library:
    https://www.geeksforgeeks.org/response-methods-python-requests/
12. ML API tutorial: https://www.youtube.com/watch?v=C82lT9cWQiA