DSCI 551 – Final Project Report

1. Introduction

ChatDB is an innovative tool designed to bridge the gap between natural language and database systems. It combines the simplicity of natural language processing (NLP) with the power of SQL and NoSQL databases, enabling users to interact with data intuitively. This project is inspired by the concept of conversational AI, like ChatGPT, but focuses on querying databases directly through natural language or structured inputs.

With ChatDB, users can upload datasets, ask for sample database queries, input queries in natural language, and receive responses seamlessly. By supporting MySQL and MongoDB, ChatDB ensures versatility in handling both structured and unstructured data. The project aims to serve as an educational and functional tool, empowering users with minimal database knowledge to extract meaningful insights.

Key Objectives:

1. Support interactive database operations through a user-friendly UI.
2. Provide sample query suggestions and assist in data exploration
3. Translate natural language into SQL/NoSQL queries, execute them and display results.

ChatDB stands out for its utilization of regex to detect complex patterns in natural language and its dynamic query generation capabilities, offering a robust solution for database interactions. It serves students, educators, and professionals looking for a hands-on approach to understanding and utilizing database systems.

2. Planned Implementation

The ChatDB project was planned to be developed in stages, ensuring a structured approach to achieving its objectives. Below is a summary of the planned implementation phases as outlined in the project proposal:

Phase 1: Project Setup and Data Handling

1. **Dataset Selection:**
    - Focused on retail and e-commerce datasets to align with real-world use cases.
    - Example datasets include sales transactions, customer records, and product details.

2. **Database Integration:**
    - Setup of MySQL for structured data and MongoDB for NoSQL data.

- Established database integration with python code using PyMongo and SQLAlchemy for seamless querying

Phase 2: Backend and Query generation / translation development

1. **Query Generation / Translation:**
    - Utilized regex to detect patterns in natural language that are equivalent to specific SQL / NoSQL constructs
    - Sample queries and SQL translation to be handled dynamically by making use of the detected patterns and metadata of the datasets being uploaded to populate predefined templates

2. **Flask Backend Implementation:**
    - Designed to act as an orchestration ground that communicates with the front end, database layer and query translation / generation layer
    - Designed routes for dataset upload, query processing, and result retrieval.

Phase 3: Frontend Development

1. **HTML User Interface:**
    - Built an interactive frontend to support dataset uploads, query inputs, and result displays.
    - Ensured responsiveness and user-friendliness.

Phase 4: Testing and Validation

1. **Testing Queries:**
    - Verified accuracy of SQL and MongoDB queries that were being generated.
    - Tested with multiple datasets for robustness.

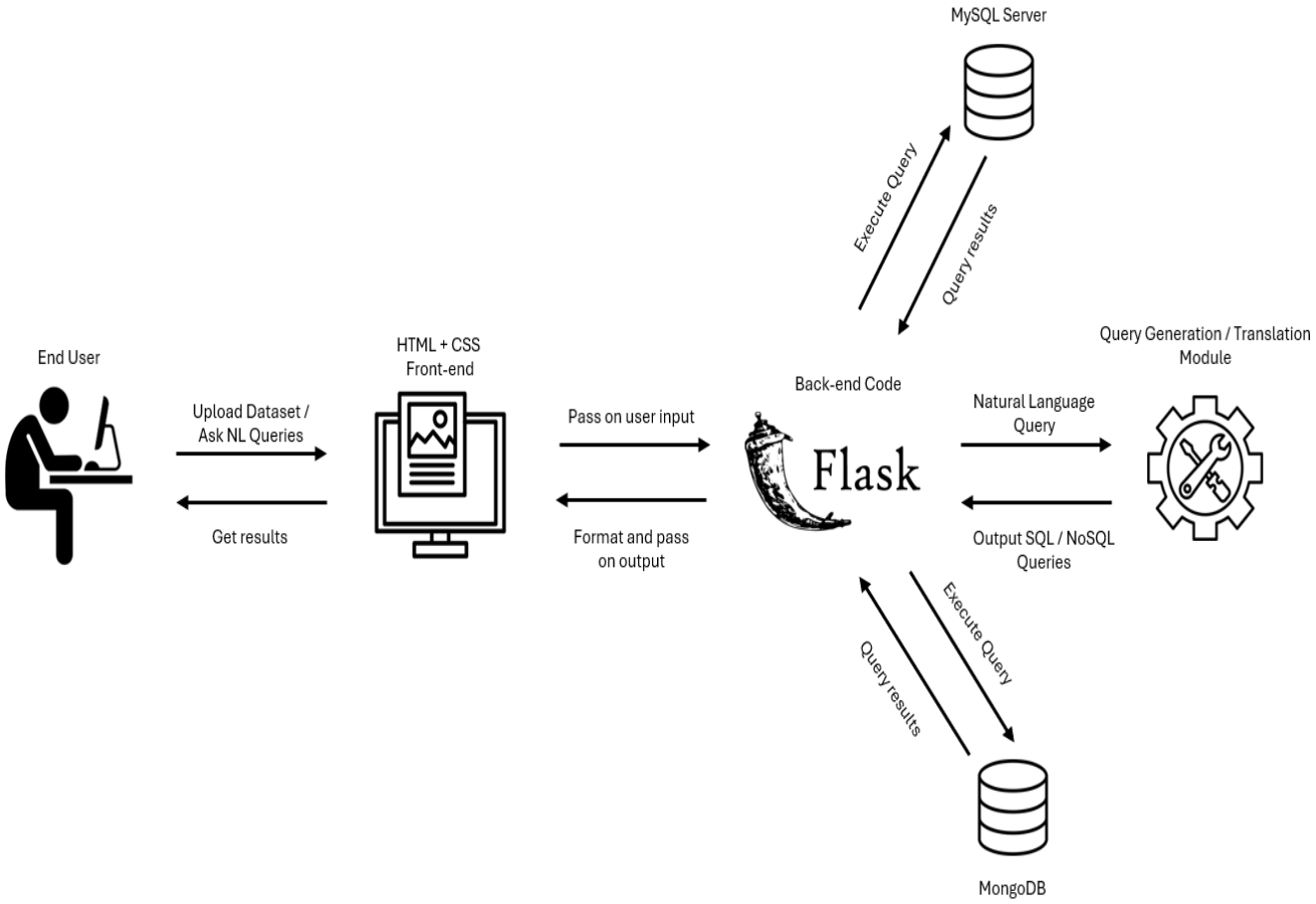
2. **Feedback Integration:**
    - Incorporated feedback to enhance user experience and system performance.

This structured approach ensured that all functionalities were implemented efficiently while maintaining focus on user-centric design and reliable database interaction.

3. Architecture Design

3.1 Flow Diagram

The architecture of the ChatDB project is organized into distinct layers to ensure modularity, scalability, and clarity of operations. Below is the flow diagram and its description.



3.2 Description of Components:

1. **Frontend Layer**:
    - Provides a user interface to interact with ChatDB. Built using HTML and Bootstrap for simplicity and responsiveness.
    - Collects inputs like database type (SQL or NoSQL), paths of datasets to be uploaded and natural language questions.
    - Displays results such as sample data / queries, translated queries, and their execution results.
2. **Backend Layer**:
    - Developed in Flask, the backend receives user inputs from the front end and passes on output data / queries to be displayed to the user.
    - Responsible for communicating with sample query generation, query translation and query execution modules.
3. **Database Layer**:
    - Used to store user datasets, execute translated queries and fetch query results.

- Switches between SQL / No SQL databases to store data and perform database operations based on the user's input in the front end

4. **Query Generation / Translation Layer**:
    - Contains helper functions that processes the input natural language query from the user and generate sample queries / translate to executable database queries
    - The layer makes use of the metadata of user uploaded datasets to aid processing of user's NL queries
    - Also has functions that aid in detecting specific language patterns within user queries so that query translation is accurate

4. Implementation

4.1 Functionalities

1. **Dataset upload and processing:**
    - Users could upload CSV datasets via the frontend interface. Based on the type of database selected, the datasets are formatted accordingly and pushed to MySQL Server / MongoDB databases
    - Schema identification and sample data previews ensured user familiarity with the dataset structure.
    - The backend also automatically processed these datasets to extract metadata that facilitated accurate query generation and translation:
        - **Attributes (Categorical Variables):** E.g., product categories or regions.
        - **Measures (Numerical Variables):** E.g., sales figures or transaction amounts.
        - **Unique Elements:** Used for generating query samples and validation.

2. **Sample query generation:**
    - Users could ask ChatDB to give sample queries using specific constructs and the back-end code would generate a random sample dynamically by making use of extracted metadata and pre-made templates
    - If asked for sample queries without any specific construct, the code would select a random sample from all available constructs and display them
    - Supports most common MySQL constructs like Where, having, group by, order by, select, limit, offset, min, max, avg, sun, etc. and MongoDB commands like project, match, limit, skip, sort, group etc.

3. **Natural Language to SQL/NoSQL Translation**
    - User entered natural language queries are parsed and translated into SQL / NoSQL dynamically. The translated queries are then executed and the results displayed to the user
    - Query translation module makes use of regex to recognize specific patterns within the natural language query, finds the columns involved and fits them into templates to form equivalent SQL/NoSQL queries.
    - Metadata helps in accurately selecting the columns required for the translation from natural language input
    - The translation function supports translation for the following operations
        1. **Aggregations:** `SUM`, `AVG`, `MIN`, `MAX`, and `COUNT`.
        2. **Filtering:** `WHERE` conditions for non-aggregates and HAVING for aggregates.
        3. **Sorting:** `ORDER BY` clauses.
        4. **Limiting & offset:** Results limited with `LIMIT`, OFFSET or `SKIP`.
        5. **Grouping:** GROUP BY and advanced aggregation pipelines.
        6. **Joins:** Queries involving attributes & measure from multiple tables

4.2 Tech Stack & Implementation details

1. **Frontend implementation**
    - A clear and simple user interface provided real-time feedback and error messages
    - **HTML** is used to define the structure of the user interface. It provides the basic skeleton for the front-end
    - **CSS** via Bootstrap 5.3 library is used for providing styling. This framework simplifies UI design and makes the front-end responsive with pre-defined classes for layout, typography, forms, tables, and buttons.
    - **JavaScript** is utilized for front-end logic and asynchronous requests using the fetch API. Provides dynamic interaction and facilitates communication with the Flask backend through AJAX calls
2. **Backend Implementation**
    - **Flask** was used to develop a lightweight and flexible backend coded in python, chosen for its simplicity, minimalism, and ease of integration with other components
    - Routes were created for dataset uploads (/load_datasets) and for query processing (/process_query). The routes made use of different helper functions

to insert the datasets onto databases, format and push sample data, sample queries, translated user queries and query execution results to the front-end

3. **Database Integration:**
    - **SQLAlchemy** handled MySQL server integration efficiently. It facilitated the execution of translated SQL queries and fetched the results.
    - The execute_sql_query() function ensured proper error handling, fetching and formatting of the results for display
    - **PyMongo** managed querying of MongoDB using dynamic aggregation pipelines. The function execute_mongo_query() ensured proper handling of MongoDB querying like its SQL counterpart
4. **Sample Query generation:**
    - Sample query generation was implemented by making **use of pre-defined templates** for each type that were **populated using a random selection of the attributes, measure and unique elements** identified during metadata extraction
    - Ex: for 'group by' one of the templates used is

f'Select {select_columns_text}, {selected_agg_function}({selected_agg_column}) as {selected_agg_column}_{selected_agg_function} from {user_dataset} group by {select_columns_text}'

    - Each of the variables above is randomly selected and populated onto the template to create a sample query on the fly
5. **Query Translation:**
    - The input natural language query is first passed through a series of pattern matching functions that use **regex** to split the input into multiple parts like where part, select part, group by part, match part, project part etc.
    - Tokens in each individual part is then compared vs the metadata available and fit into templates to generate sub sections of the final output SQL / NoSQL query
    - From and Join part is then formed based on the columns present in all translated parts so far
    - The sub sections created are then stitched together in the right order to form the final translated query
    - Ex: for the user input 'Give me average of quantity by saledate when saledate is greater than 2021-01-01' would be split and translated as shown in the table below

Section type	Split of Natural Language	Translated SQL Part
Select + Group by	*('Give me average of quantity', 'by saledate')*	*Select saledate, Avg(quantity) as avg_quantity group by saledate*
Where	*When saledate is greater than 2021-01-01'*	*Where saledate > '2021-01-01'*
From / Join	*** Based on columns present in previous parts****	*From sales*

- The translated SQL parts are then arranged to form the complete translated query as follows

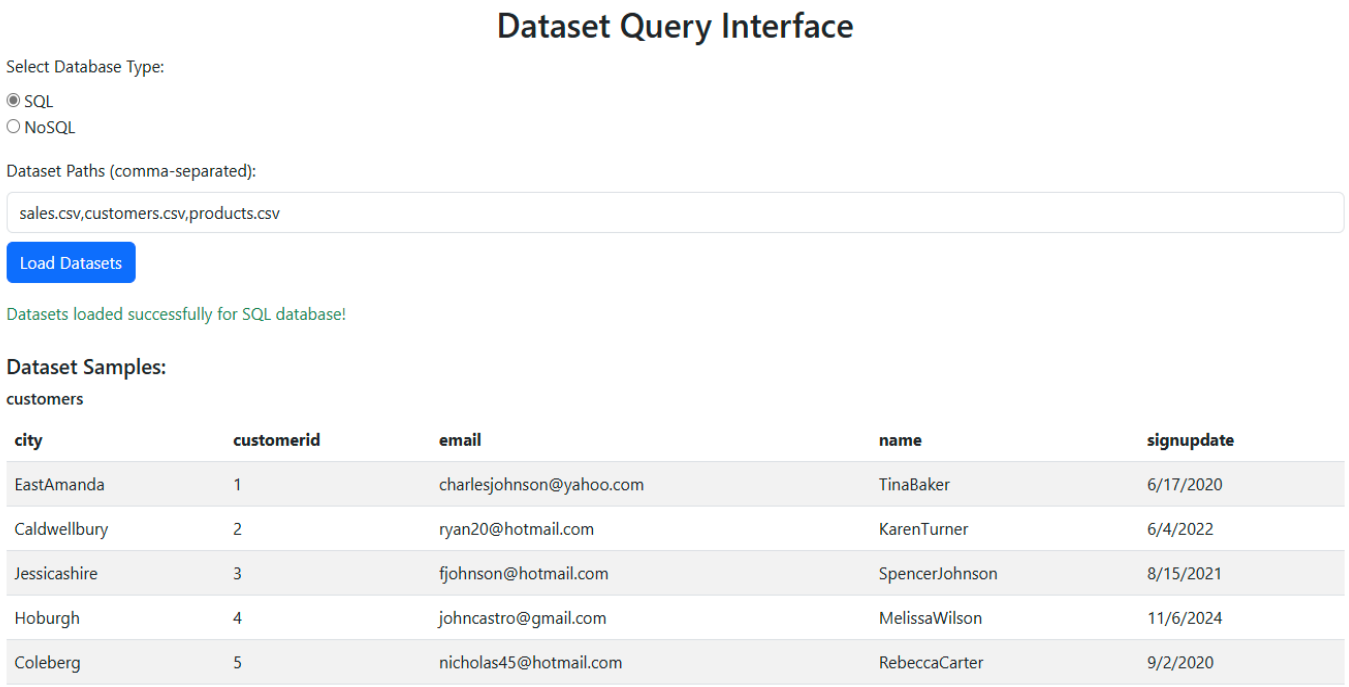
Select saledate, Avg(quantity) as avg_quantity from sales where saledate > '2021-01-01' group by saledate

- MongoDB query translation also makes use of a similar process of pattern detection, template population and part stitching as the SQL counterpart

4.3. Implementation Screenshots

1. Dataset Upload & sample data display:

MySQL



products

category	price	productid	productname	stockquantity
Toys	442.34	1	Economy Tool	93
Books	46.28	2	Market Item	100
Electronics	188.55	3	Information Tool	78
Toys	290.17	4	Adult Gadget	80
Home	429.67	5	If Gadget	53

sales

customerid	productid	quantity	saledate	saleid	totalamount
2	13	2	2024-06-24	1	709.66
10	5	2	2023-06-18	2	859.34
5	13	3	2024-07-30	3	1064.49
2	8	2	2024-09-14	4	219.9
6	5	5	2024-01-04	5	2148.35

MongoDB

Dataset Query Interface

Select Database Type:

○ SQL
◉ NoSQL

Dataset Paths (comma-separated):

sales.csv,customers.csv,products.csv

Load Datasets

Datasets loaded successfully for NoSQL database!

Dataset Samples:

customers


```
[
  {
    "city": "EastAmanda",
    "customerid": 1,
    "email": "charlesjohnson@yahoo.com",
    "name": "TinaBaker",
    "signupdate": "6/17/2020"
  },
  {
    "city": "Caldwellbury",
    "customerid": 2,
    "email": "ryan20@hotmail.com",
    "name": "KarenTurner",
    "signupdate": "6/4/2022"
  },
  {
    "city": "Jessicashire",
    "customerid": 3,
    "email": "fjohnson@hotmail.com",
    "name": "SpencerJohnson",
    "signupdate": "8/15/2021"
  },
  {
    "city": "Hoburgh",
    "customerid": 4,
    "email": "johncastro@gmail.com",
    "name": "MelissaWilson",
    "signupdate": "11/6/2024"
  },
  {
    "city": "Coleberg",
    "customerid": 5,
    "email": "nicholas45@hotmail.com",
    "name": "RebeccaCarter",
    "signupdate": "9/2/2020"
  }
]
```

products


```
[
  {
    "category": "Toys",
    "price": 442.34,
    "productid": 1,
    "productname": "Economy Tool",
    "stockquantity": 93
  },
  {
    "category": "Books",
    "price": 46.28,
    "productid": 2,
    "productname": "Market Item",
    "stockquantity": 100
  },
  {
    "category": "Electronics",
    "price": 188.55,
    "productid": 3,
    "productname": "Information Tool",
    "stockquantity": 78
  },
  {
    "category": "Toys",
    "price": 290.17,
    "productid": 4,
    "productname": "Adult Gadget",
    "stockquantity": 80
  },
  {
    "category": "Home",
    "price": 429.67,
    "productid": 5,
    "productname": "If Gadget",
    "stockquantity": 53
  }
]
```


sales


```
[
  {
    "customerid": 2,
    "productid": 13,
    "quantity": 2,
    "saledate": "2024-06-24",
    "saleid": 1,
    "totalamount": 709.66
  },
  {
    "customerid": 10,
    "productid": 5,
    "quantity": 2,
    "saledate": "2023-06-18",
    "saleid": 2,
    "totalamount": 859.34
  },
  {
    "customerid": 5,
    "productid": 13,
    "quantity": 3,
    "saledate": "2024-07-30",
    "saleid": 3,
    "totalamount": 1064.49
  },
  {
    "customerid": 2,
    "productid": 8,
    "quantity": 2,
    "saledate": "2024-09-14",
    "saleid": 4,
    "totalamount": 219.9
  },
  {
    "customerid": 6,
    "productid": 5,
    "quantity": 5,
    "saledate": "2024-01-04",
    "saleid": 5,
    "totalamount": 2148.35
  }
]
```


2. Sample Query Generation:

MySQL

Enter Natural Language Query:

Give me sample queries with group by

Submit Query

Sample Queries:

- Select productid,category from products group by productid,category
- Select productid,productname from products group by productid,productname
- Select productname,productid from products group by productname,productid
- Select customerid,saledate from sales group by customerid,saledate
- Select customerid,productid,saledate,Avg(quantity) as quantity_Avg from sales group by customerid,productid,saledate

Enter Natural Language Query:

Give me sample queries with group by

Submit Query

Sample Queries:

- Select saleid,customerid,Avg(quantity) as quantity_Avg from sales group by saleid,customerid
- Select category from products group by category
- Select saleid,customerid from sales group by saleid,customerid
- Select category from products group by category
- Select productid,saledate,Sum(quantity) as quantity_Sum from sales group by productid,saledate

Enter Natural Language Query:

Give me sample queries with having

Submit Query

Sample Queries:

- Select productid,count(*) as cnt from products group by productid having Cnt > 337
- Select saleid,customerid,Max(totalamount) as totalamount_Max from sales group by saleid,customerid having totalamount_Max >= 307
- Select productid,count(*) as cnt from products group by productid having Cnt >= 408
- Select customerid,Min(quantity) as quantity_Min from sales group by customerid having quantity_Min <= 680
- Select customerid,saledate,count(*) as cnt from sales group by customerid,saledate having Cnt > 404

MongoDB

Enter Natural Language Query:

Give me sample queries with match

Submit Query

Sample Queries:

- db.products.aggregate([{'$match': {'productname': 'One Gadget'}}])
- db.products.aggregate([{'$match': {'productid': 7, 'category': 'Electronics'}}])
- db.sales.aggregate([{'$match': {'productid': 11, 'customerid': 8, 'saleid': 42}}])
- db.products.aggregate([{'$match': {'productid': 6, 'productname': 'Prepare Item'}}])
- db.products.aggregate([{'$match': {'category': 'Clothing'}}])

Enter Natural Language Query:

Give me sample queries with match

Submit Query

Sample Queries:

- db.products.aggregate([{'$match': {'productname': 'Economy Tool', 'productid': 2}}])
- db.sales.aggregate([{'$match': {'saledate': '2023-10-28', 'productid': 9, 'customerid': 9}}])
- db.products.aggregate([{'$match': {'productname': 'Economy Tool'}}])
- db.sales.aggregate([{'$match': {'saledate': '2023-06-03', 'saleid': 30, 'productid': 15, 'customerid': 5}}])
- db.sales.aggregate([{'$match': {'saledate': '2023-03-07'}}])

Enter Natural Language Query:

Give me sample queries with project

Submit Query

Sample Queries:

- db.products.aggregate([{'$project': {'productid_new': '$productid', 'category_new': '$category', 'price_new': '$price', 'stockquantity_new': '$stockquantity'}}])
- db.sales.aggregate([{'$project': {'quantity_new': '$quantity'}}])
- db.sales.aggregate([{'$project': {'totalamount_new': '$totalamount'}}])
- db.sales.aggregate([{'$project': {'saleid_new': '$saleid'}}])
- db.products.aggregate([{'$project': {'stockquantity_new': '$stockquantity', 'price_new': '$price'}}])

Enter Natural Language Query:

Give me sample queries with group

Submit Query

Sample Queries:

- db.sales.aggregate([{'$group': {'_id': '$customerid', 'sum_quantity': {'$sum': '$quantity'}}}, {'$sort': {'_id': 1}}])
- db.sales.aggregate([{'$group': {'_id': '$productid', 'sum_quantity': {'$sum': '$quantity'}}}, {'$sort': {'_id': 1}}])
- db.products.aggregate([{'$group': {'_id': '$productid', 'sum_stockquantity': {'$sum': '$stockquantity'}}}, {'$sort': {'_id': 1}}])
- db.products.aggregate([{'$group': {'_id': '$productname', 'sum_price': {'$sum': '$price'}}}, {'$sort': {'_id': 1}}])
- db.products.aggregate([{'$group': {'_id': '$category', 'sum_stockquantity': {'$sum': '$stockquantity'}}}, {'$sort': {'_id': 1}}])

3. Query Translation and execution:

MySQL

Enter Natural Language Query:

Give me average of quantity by saledate and category when saledate is between 2021-01-01 and 2023-12-31 and average quantity is greater than 3

Submit Query

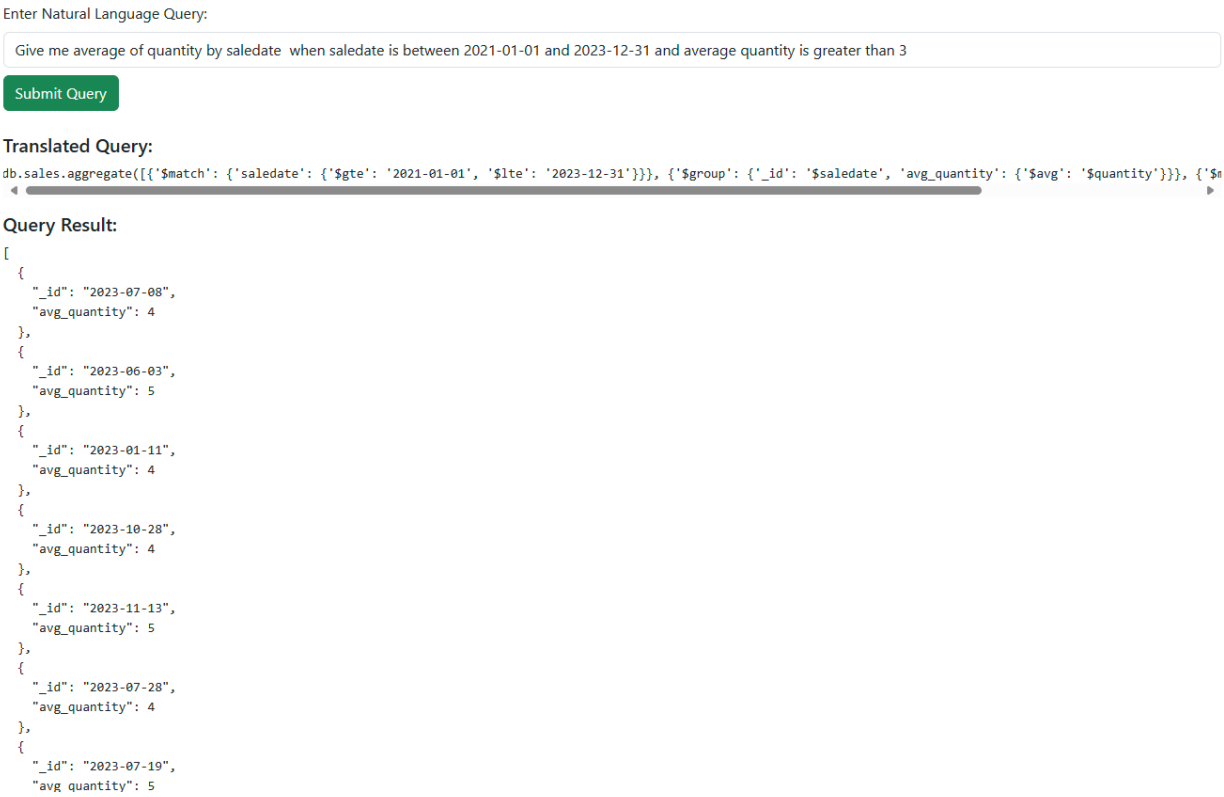
Translated Query:

```
Select saledate,category,Avg(quantity) as avg_quantity  from sales join products on sales.productid = products.productid  where saledate between '2021-01-01' and '2023-
```

Query Result:

avg_quantity	category	saledate
4.0000	Electronics	2023-10-28
4.0000	Toys	2023-03-07
4.0000	Toys	2023-07-08
5.0000	Toys	2023-07-19
4.0000	Toys	2023-07-28
5.0000	Toys	2023-11-13
4.0000	Books	2023-01-11
5.0000	Electronics	2023-01-03
5.0000	Electronics	2023-12-12

MongoDB



5. Learning Outcomes

The ChatDB project offered a wealth of learning opportunities, enhancing our technical expertise and collaborative abilities.

5.1 Technical Skills

- **Database Integration:**
    - Gained hands-on experience in integrating both relational (MySQL) and non-relational (MongoDB) databases to python code.
- **Natural Language Processing:**
    - Developed deep understanding of SQL / No SQL query formats and their potential natural language counterparts
    - Learnt to generate sample queries dynamically using user uploaded dataset's metadata
    - Mastered pattern matching using regex and query translation using python

- **Full-Stack Development:**
    - Improved skills in developing an full stack app using Flask, HTML and JavaScript and communicating effectively and effortlessly

5.2 Team Collaboration

- Fostered effective communication and task distribution among team members.
- Learned to coordinate through Git for version control and codebase management.

5.3. Challenges Faced

The ChatDB project encountered several challenges that tested our skills and adaptability. Below are the key obstacles and strategies employed:

1. Diverse Dataset Compatibility

Challenge: Handling datasets with varying schemas and types of data complicated query generation.
Solution:

- Implemented dynamic schema detection to adapt to different datasets.
- Designed flexible column mapping algorithms to accommodate diverse dataset attributes and measures.

2. Query Translation Complexity

Challenge: Translating natural language queries into precise SQL/NoSQL queries was difficult due to the number of query types involved
Solution:

- Developed a pattern-matching system to identify common query structures.
- Came up with robust templates that could be readily plugged with attribute and measure names to form executable SQL query

3. Frontend-Backend Communication

Challenge: Achieving smooth communication between the frontend and backend for query processing and result display.
Solution:

- Implemented usage of Flask for scalable back-end processing and HTML + JS for responsive frontend

6. Individual Contributions

The ChatDB project was a collaborative effort, utilizing the strengths of each team member. Below is a summary of their roles and contributions:

6.1 Damien Hailson

Role: Natural language processing, sample query generation and query translation

Key Contributions:

- Developed templates and logics used for dynamic Sample query generation
- Came up with the regex patterns used to detect various components within Natural language input
- Developed logic for translation of detected Natural language parts into SQL / NoSQL counterparts and generation of final translated output

6.2 Tushar Kaushik

Role: Backend Development and Database Integration

Key Contributions:

- Developed Flask backend routes to handle dataset uploads, query processing, and result retrieval, ensuring seamless communication between the UI and query translation modules.
- Integrated MySQL and MongoDB databases using SQLAlchemy and PyMongo, implementing efficient error handling for query execution.
  Designed helper functions like execute_sql_query() and execute_mongo_query() to manage database operations, ensuring reliability and consistency.

6.3 Aditya Chunduri

Role: Frontend Development and User Interface Design

Key Contributions:

- Built the HTML and JavaScript-based frontend to support real-time query input, dataset upload, and results display, ensuring an interactive and user-friendly experience.
- Utilized Bootstrap 5.3 to design a responsive layout, improving usability across devices and screens.
- Implemented dynamic feedback mechanisms and AJAX calls for smooth communication between the frontend and Flask backend.

7. Conclusion

The ChatDB project successfully demonstrates the potential of combining natural language processing with database query systems to create an intuitive and efficient interface for data exploration. The integration of MySQL and MongoDB, along with the Flask backend and HTML frontend, provides a robust framework capable of handling diverse datasets and query types. Through extensive testing and implementation, the project achieves its goal of simplifying data querying for users with minimal technical expertise.

Key achievements include:

- A seamless integration of SQL and NoSQL databases.
- A user-friendly interface supporting dynamic dataset uploads and query results.
- Reliable query generation for both simple and complex natural language inputs.

8. Future Scope

Building upon the success of ChatDB, there are several areas for potential enhancement:

1. **Advanced Query Support:**
    - Expand support for nested queries and advanced SQL features like subqueries.
    - Enhance MongoDB query generation to include complex aggregation pipelines like joins.
2. **Cloud Integration:**
    - Deploy the system on cloud platforms to enable scalability and accessibility for enterprise use.
3. **Analytics and Visualization:**
    - Integrate data visualization tools to present query results graphically, improving insights and usability.

By addressing these areas, ChatDB can evolve into a comprehensive and versatile data querying tool, catering to a wide range of users and industries.

Code Repository Links

Google Drive:
https://drive.google.com/drive/folders/1qD82oAqTAdmrEoCRu3z2DPu4sA7Tpbl0?usp=sharing

Github:
https://github.com/Damien-Hailson/DSCI-551-Project-ChatDB-46.git