**Final Project Report**

Jiawei Qin, Jietong Lei, Kaidi Jiang

1. **Introduction**

Our comprehensive project aims to deliver a versatile Database Management System that encompasses both NoSQL and Relational databases, coupled with a user-friendly web interface.

We started by tackling big challenges like handling large data files and figuring out how to write complex queries in Python. Our main goal was to simplify data management. We also picked a good dataset from Kaggle to work with, which helped us learn a lot about how different pieces of data can be connected.

We created and saved the database as a folder, and the table as a file. To solve the problem about restricted main memory size, we splited data into chunks and stored them in the table file, which improves the efficiency of subsequent data processing.

Our database management system has its own query languages, and we also developed an interactive command line interface, along with some other pages which allow user to interact with database system. This project is really about making it easier for people to manage and understand data, even if they're not database experts.

1. **Planned Implementation**

The objective was to develop a lightweight relational database management system supporting basic Create, Read, Update, Delete (CRUD) operations. The key goal was to provide a straightforward interface for users to interact with a database easily.

1. **Implementation - NoSQLDatabase**

The NoSQL database system has its own programming algorithms for CRUD (Create, Read, Update, Delete) function and other query function such as ordering, group by, and join in db1.py file.

**Functionalities**

1.Create\_database: create a folder with database\_name.

2.Create\_table: create a file with table\_name and schema are the columns name of the table.

3.Delete\_table: delete the file with table\_name.

4.Load\_existing\_database: load the existing database with database\_name and find the folder.

5.Load\_data: load all the table file from the database folder.

6.Save\_data: save the table into JSON file.

7.Insert\_single\_row: insert data into table with table\_name, modify data as chunks.

8.Create\_condition: create a condition for other query function.

9.Delete\_data: delete entire row from table with table\_name which matches the condition.

10.Update\_data: update entire row from table with table\_name which matches the condition.

11.Order\_data: order the specified table based on the specified column with old\_table\_name, then create a new table with new\_table\_name to store the result.

12.Groupby: group the specified table based on the specified column with old\_table\_name, then create a new table with new\_table\_name to store the result.

13.Sum/count/avg/max/min: calculate the sum/count/avg/max/min of the specified column from old\_table\_name, then create a new table with new\_table\_name to store the result.

14.Join: join two tables based on the common key and create a new table new\_table\_name to store the result.

15.Show\_data: show the specified table data with table\_name.

These above functions show the basic database functionality with file system. We also designed a command line interface file command.py that allows the users to modify the data through database system. If the command line starts with specified key, command.py will start difference operation based on the key. For example, If the command line starts with create\_database, this code will create a database with given name. If the command line starts with show\_data, it will ask users how many rows they need, which is the limit function. Then the system will ask users if they need condition, which is filter function. Then the system will ask users if they need show specified columns, which is projection function. If the command line starts with group by, it will group the data based on the specified column, and then the system will ask users if they need aggregation function. If users want to quit the system, they can just use exit command. Below are some implementation screenshots.

**Tech Stack**

The project primarily uses Python. Key libraries include:

1. os for file and directory operations
2. csv for reading and writing CSV files
3. prompt\_toolkit for the command-line interface

**Implementation Screenshots**

Create database named sample:

A close up of a word

Description automatically generated

Show table person:

A close-up of a computer code

Description automatically generated

Find the min age in table person:

A close-up of a white background

Description automatically generated

Groupby table person by name and find max age:

A white background with black text

Description automatically generated

Join table person and book with person.name and book.author:

A white background with black text

Description automatically generated

**4.Implementation - RelationalDatabase**

The implementation consists of two main components:

**MyRelationalDatabase Class:** This is a custom Python class we made. It acts like a small relational database. It's got functions for making new databases and tables, and for adding, changing, or removing data. We made sure it could do all the usual stuff you'd expect in a database, like sorting and finding specific bits of data.

**Command-Line Interface Application:** This is the part where users interact with our database. We used the prompt\_toolkit library to make it work. It takes in commands from users and then uses the MyRelationalDatabase class to do what the user wants, like adding new data or showing what's in the database. This way, users can talk to our database directly through their command line, making it super user-friendly.

**Functionalities**

The system supports a range of functionalities:

1. Creating and deleting databases and tables
2. Loading existing databases
3. Inserting data from CSV files and single data entries
4. Deleting and updating data based on conditions
5. Displaying data with options for limiting rows, applying conditions, and specifying columns
6. Performing aggregation operations like sum, count, average, max, and min
7. Ordering and joining data

**Tech Stack**

The project primarily uses Python. Key libraries include:

1.os for file and directory operations

2.csv for reading and writing CSV files

3.prompt\_toolkit for the command-line interface

**Implementation Screenshots**

文本

描述已自动生成文本, 信件

描述已自动生成

文本

描述已自动生成文本

描述已自动生成文本, 信件

描述已自动生成These are some screenshots showing create table, sorting, filtering, joining and counting command running on our model.

**5.Implementation - WebInterface**

We have created several web pages for our application. The homepage allows users to choose different databases. Based on their selection, we use different classes and databases to process subsequent user commands. Once the user selects a database, they are redirected to our database interface main page, which features a primary command input box. We handle user-input commands on the backend, matching the first command keyword to relevant command processing functions. The split values are then used as input, and the results from the database class are returned.

We provide clear feedback to users by offering success or error messages, keeping them informed about the status of their commands. In the case of successful execution, users are directed to a dedicated results page, where they can view the outcome of their command. For commands resulting in data stored in file format, users have the option to use commands like "show data" to visualize the results directly or navigate to their local files to explore detailed information about their database.

**Functionalities**

1. homepage for switching between 2 databases
2. database interface page for inputing commands
3. result page for showing the results (only for show\_data command)

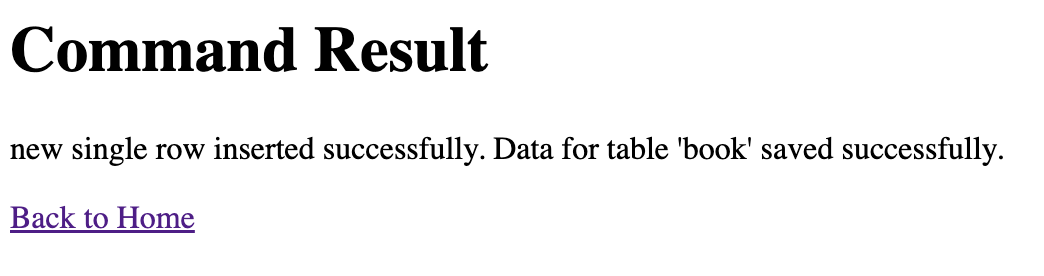
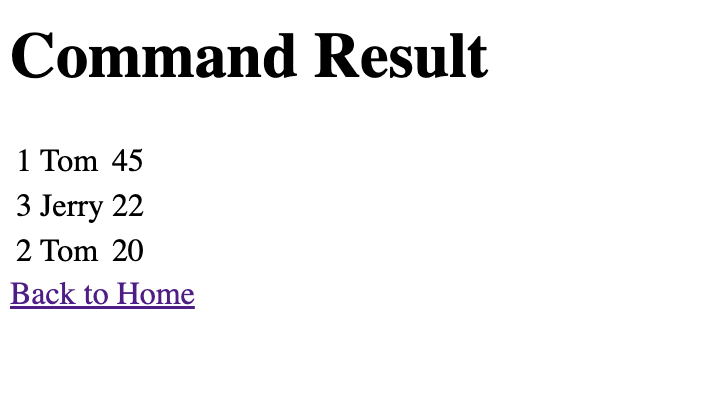
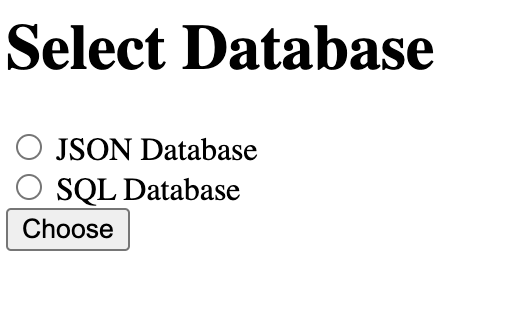
**Tech Stack**

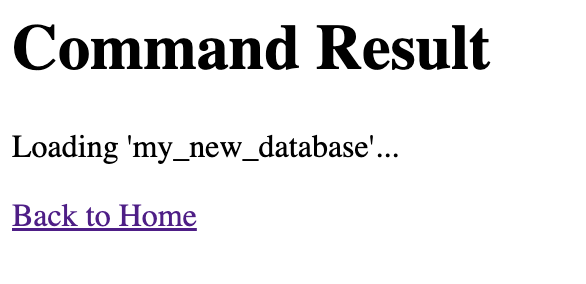
Our web application leverages the Flask framework, a lightweight and efficient Python web framework, to build a user-friendly interface for interacting with our NoSQL and relational databases.

The backend of the application is powered by Python, and we use Flask to handle HTTP requests and responses. For the NoSQL database, we use the db1 class, and for the relational database, we employ the MyRelationalDatabase class. The application's logic is encapsulated in the cc and cc\_sql classes, which process user commands and interact with the respective databases.

The web pages are rendered using HTML templates and styled with CSS. The entire application is hosted and run locally, providing users with an accessible and responsive platform for efficient database management.

**Implementation Screenshots**





**6.Learning Outcomes**

We gained a deeper understanding of nosql and relational database, got to think the inner logic of every command operation, learned how to build a database from the ground up, which was a big task. We also got to grips with CRUD operations, which are the basics of adding, reading, updating, and deleting data in Python applications. This meant we had to dive deep into Python libraries, not just for managing files but also for creating user interfaces. Build the user interface is also a brand new experiement for us, we are happy to learn and practice on building connections from the web and the database we designed. It was a great way to see how different pieces of Python come together to make something functional and user-friendly.

**7.Challenges:**

We faced some tough challenges in this project:

**Handling Data Types in CSV Imports:** One of the trickiest parts was making sure that when we imported data from CSV files, we kept the data types right. It was crucial for our database to understand and differentiate between numbers, text, and other types of data.

**Efficient Data Storage and Retrieval:** We also had to figure out how to store and retrieve data efficiently. This was important to make sure our database was fast and reliable, even when dealing with large amounts of data.

**User-Friendly Command-Line Interface:** Lastly, creating an easy-to-use command-line interface was a challenge. We wanted to make sure that even people who aren't tech-savvy could use our database without getting lost or confused. This meant designing a clear and simple interface that anyone could navigate.

**Creation of composite functions:** Since all the aggregation function in our database system create a new table with a new name, it is difficult to use the new table in other function. Therefore, we merge different functions together through a question and answer format.

**Building web interface:** We faced a challenge while transitioning the group by command from the terminal to the web interface. In our initial implementation, we used 'prompt' in both databases, but adapting it to the web proved to be a complex task. The intricacies of incorporating prompts into a web environment presented difficulties in designing the inner logic of the database. Moving forward, we acknowledge the need for a more refined and intuitive solution to overcome this challenge. We plan to thoroughly review and improve the logic to ensure a seamless and effective implementation in the future.

**8.Individual Contribution**

Jiawei Qin: Designing and implementing Nosql Database System.

Jietong Lei: Designing and implementing Relational Database System.

Kaidi Jiang: Designing frontend user interface.

**9.Conclusion**

We built a basic nosql and a relational database management system that's easy to use. It's got a simple command-line interface, perfect for people who aren't database experts. This project was a great way to show how basic database operations work. It's a solid foundation for anyone interested in database systems.

**10.Future Scope**

We've got big plans for the future of this project. We want to add more advanced query options, so users can do more with their data. We're also thinking about building a graphical user interface. This would make our database even easier to use. Plus, we're planning to boost our security features to keep data safe and sound.