**Voter Registration: Database Maintenance**

**Introduction**

The Nigerian general elections were held on 25 February 2023. As a Nigerian a keen interest in the democratic process of my country. An issue that I found particularly interesting was the disparity between the number of registered voters and the actual voter turnout. With a record of 93.5 million registered voters, only 87.3 million had collected their permanent voters' cards, and less than a third of them voted. The total turnout was about 27%. This issue has been attributed to numerous reasons such as deaths not being reported, people relocating, and safety concerns Given the rigging allegations of the elections, this made me question the efficiency of the voter registration process and database management in Nigeria. To address this issue, this project aims to develop an accurate and efficient voter registration database in Nigeria and to increase voter turnout in Nigeria. This is critical in ensuring a free and fair election. Specifically, the project proposes the implementation of two algorithms: one that updates the database by removing inactive voters and another that helps voters locate their polling unit.

**Solution Approach**

Searching Database

To solve the problem of voters finding their polling unit, a web application was developed using the Flask framework, and the SQLite database that was previously generated. This application allows voters to input their voter ID, date of birth, and name, and it returns their assigned polling unit.

The search algorithm first hashes the voter ID using the SHA-256 algorithm, a very secure and efficient hash function, it then uses the hashed ID to query the database for all voter IDs associated with the hash value. If a match is found, note that for a match to be found the name and the date of birth also must be correct, the algorithm retrieves the information about the assigned polling and returns it to the user. If a match was not found, an error message is returned. The hash function is designed to run in constant time; hence the time complexity should be O (1) for the average case. However, the performance of the algorithm can be impacted by the size of the database. More than one voter ID having the same voter ID leads to a collision and increases the time complexity of the algorithm. With the use of the SHA-256 hash function, the probability of that occurring is low but not impossible.

**The code snippet for finding the polling unit is shown below:**

**Text

Description automatically generated**

The search algorithm handles the case of collisions. If there is more than one voter ID associated with the hash function, the algorithm searches through all polling units with the voter ID and finds the one assigned to the voter ID being searched. The time complexity for this case would be O(m) where m is the number of voter IDs associated with the hash function. In the worst-case scenario, all voter IDs are associated with the same hash value. In that case, the time complexity would be O(n).

Updating Database.

To solve the database maintenance issue, a database management algorithm was developed. This approach involves identifying and deleting inactive voters from the database. The algorithm creates an index on the “last\_participation\_date” column. This index uses the [B-Tree](https://www.geeksforgeeks.org/introduction-of-b-tree-2/) data structure, a self-balancing tree, which searches and sorts the database efficiently. After the index has been created, a query is run to find voters that have been inactive in the last two elections, the last eight years. With the use of the index, the time complexity of this query is O(logn). The algorithm then deletes the inactive voters from the database by looping through the inactive voters. The time complexity of this loop is O(m) where m is the number of inactive voters being deleted from the database. Therefore, the overall time complexity of O(mlogn). In the worst-case scenario, all the entries are inactive resulting in a time complexity of O(nlogn)

**The code snippet for the database management is shown below:**

**Text

Description automatically generated**

**Experiment:**

Data Creation

For this project, a mock voter registration database was created using Python and several libraries including [SQLite](https://docs.python.org/3/library/sqlite3.html), [hashlib](https://docs.python.org/3/library/hashlib.html), [uuid](https://docs.python.org/3/library/uuid.html), [Random](https://docs.python.org/3/library/random.html), and [Faker.](https://github.com/joke2k/faker) This script randomly generated 100,000 voter records with fields for the voter’s name, address, email, phone number, voter ID, polling unit ID, last participation date, date of birth, and hash ID. The voter ID and the has ID were both generated using the uiud and hashlib libraries respectively, while the polling unit ID was generated using the random library. The other fields were all generated using the Faker library.

**The code snippet for generating the mock voter data is shown below:Text

Description automatically generated**

A table was created in the voter registration database to store information concerning each polling unit including fields for name, and address city. The connection of the table to the database was established using SQLite. The Faker Library was used to generate data for 100 polling units. To make the data accessible, It was exported to a CSV file with the use of the pandas library.

**The code snippet for generating the polling unit is shown below:**

Text

Description automatically generated

The aim of this database creation was to simulate a realistic voter registration database to provide a suitable data set for the implementation of the algorithm.

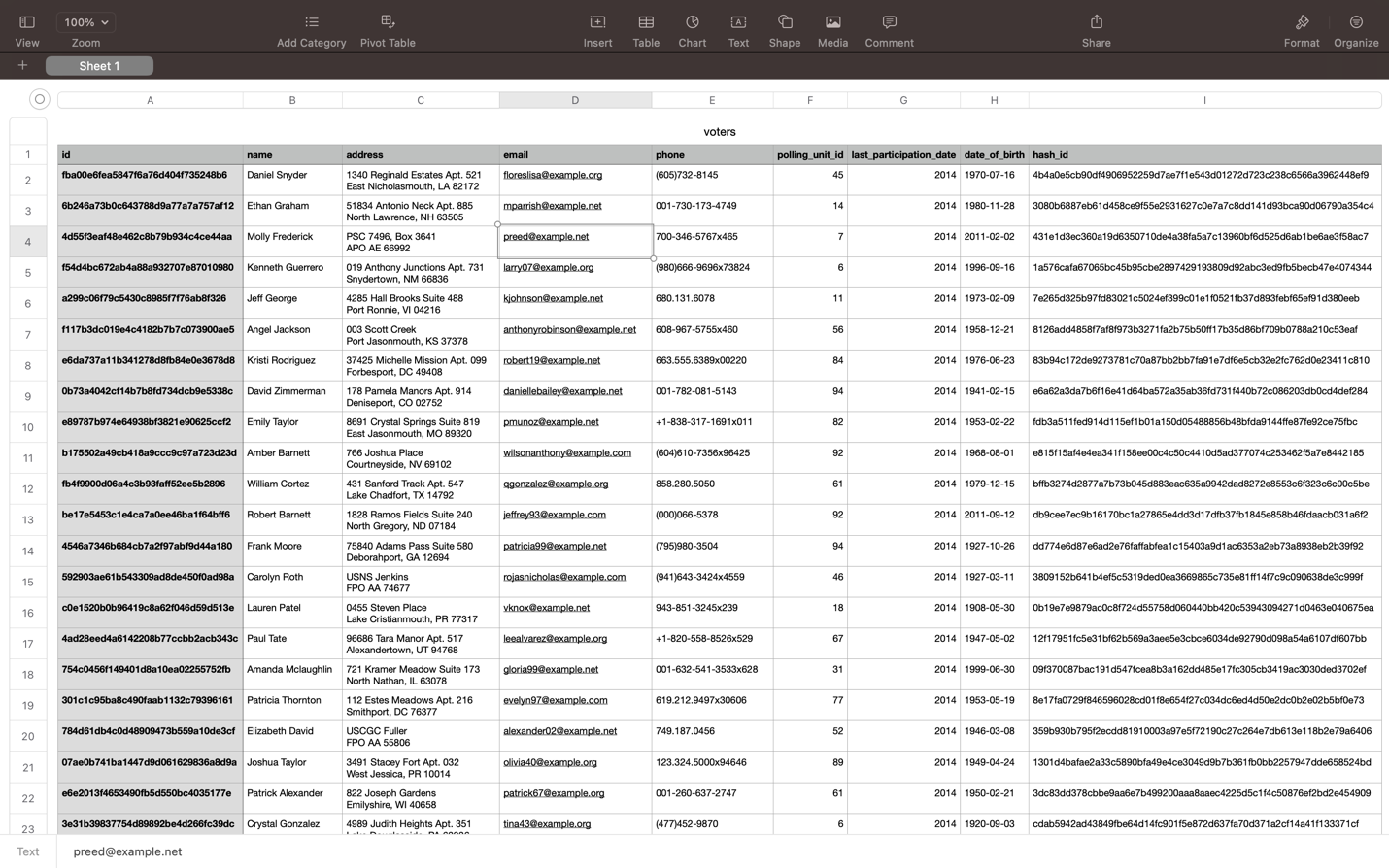
**The image below shows a snippet of the voters database:**

**Before maintenance**

A computer screen capture

Description automatically generated with medium confidence

**After maintenance**



Webpage Creation:

A simple webpage was created to mimic a voter information portal. This webpage includes a form for the voter to input their name, date of birth and voter ID. The portal is designed to emulate the process of retrieving voter information from the database. The aim of this webpage is to access the performance of the algorithm in a real-world scenario.

**An image of the portal is shown below:** Graphical user interface, application

Description automatically generated

Text

Description automatically generated

**Challenges:**

A huge challenge I face was optimizing my algorithm, specifically the data maintenance algorithm. The initial query had a time complexity of O(n). This was not scalable for a database of the size anticipated. To address this issue, I researched implementing a priority queue and came across the use of an index instead which uses a B-tree data structure. This significantly reduced the overall time complexity from O(n^2) to O(nlogn).

Another challenge I faced was connecting the database to the webpage. This would be my first attempt therefore I did some research and learnt how to use Flask to achieve this. It was hard to debug when I ran into issues with it but with patience and paying attention to details, I was able to overcome it.

One unique feature my project incorporates is the data maintenance algorithm, which identifies and removes inactive voters. This ensures that the database is up to date and provides accurate information. This can be used to provide reliable estimates of voter turnout in elections to come. While I was able to address these challenges, there were some other problems I couldn’t resolve due to time constraints, I had to limit the scope of the project. I initially planned on developing an algorithm to assign polling units to voters based on their addresses. Using fake data posed a challenge and limited my ability to do so.

**Conclusion:**

This project has allowed me to gain valuable experience in database management and web development. I discover the Flask library and learn how to use it to connect the database to a web interface. I was able to implement algorithms and data structures that were covered in class. I plan on expanding the project by incorporating real-world data, with the most emphasis on the address and polling unit fields to implement an algorithm for assigning polling units. I also plan on improving the user interface by adding more interactive features.