**1. Introduction**

The expansion of cloud services has accelerated the popularity of database-as-a-service (DBaaS) solutions, which provide scalable and dependable database services to customers. The ease of delegating database maintenance to cloud service providers, on the other hand, raises security problems, notably the potential of unauthorized access by hostile insiders. This project solves these security issues by developing, creating, and testing a Django-based secure database-as-a-service solution.

**Background:**

Users of conventional database services must balance data security with the usage of cloud-based alternatives. Encrypting data before sending it to the database improves security but complicates searching. The project's goal is to strike a balance by enabling customers to protect their data in the database while still reaping the advantages of cloud services.

**Project Goals:**

The primary goals of the project include:

* Implementing user authentication mechanisms to ensure secure access to the system.
* Incorporating a basic access control mechanism for different user groups (Group H and Group R).
* Providing query integrity protection to detect modifications or fake data.
* Enforcing data confidentiality protection for sensitive attributes.

**2. Project Setup**

**2.1 Database System Setup**

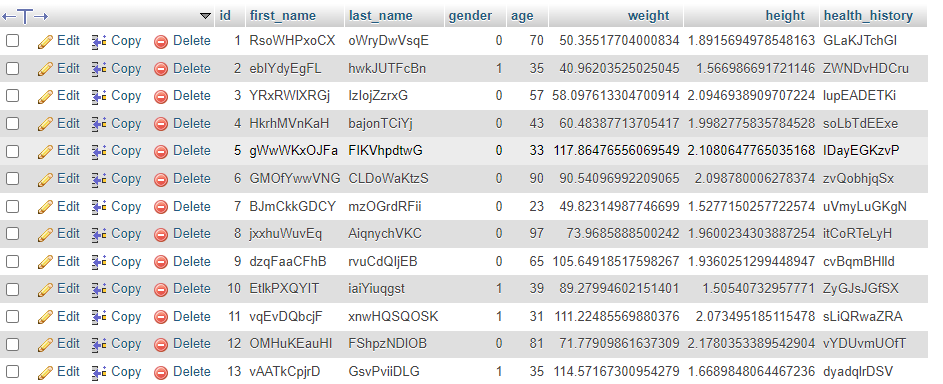
For this project, we opted to use the MySQL database as our chosen SQL database system. While acknowledging that the actual deployment may involve cloud services, for the purpose of this project, we simulated a local environment, mimicking a cloud-like setup.

**2.2 Table Creation and Data Population**

In line with the project requirements outlined in Section 4.1, we initiated the setup by creating a database table that captures healthcare information. The table structure includes the following fields:

* First name (string)
* Last name (string)
* Gender (boolean)
* Age (integer)
* Weight (floating)
* Height (floating)
* Health history (text)

The population of the table was accomplished by generating and inserting a minimum of 100 data items. This step was essential for testing and validating the implemented security features in a realistic scenario. By populating the table with diverse data, we aimed to assess the robustness of our system against various user interactions and potential security threats.

Top of Form

**3. Security Features**

**3.1 User Authentication**

*Implementation Details:* User authentication was implemented using a custom mechanism to ensure the security of the authentication process. We employed the traditional username/password combination for user authentication. Notably, the original password was not stored in the cloud; instead, a secure representation, such as a hash, was stored to enhance security.

**3.2 Basic Access Control**

*Design and Implementation:* The basic access control mechanism was designed to accommodate two user groups - Group H and Group R. Group H has access to all fields, while Group R can access all fields except for first name and last name. This was achieved by incorporating user roles into the system, with specific permissions assigned to each group. Users from both groups can query existing data, but the returned data items adhere to the specified access restrictions.

**3.3 Basic Query Integrity Protection**

*Implementation Details:* Query integrity protection was addressed by focusing on two aspects:

* *Single Data Item Integrity:* Users, regardless of their group, are equipped to detect modifications or fake data in a returned data item.
* *Query Completeness:* The system allows users to detect the removal of one or more data items from a query result. This is achieved with a certain level of probability, ensuring that users can identify potential data manipulation.

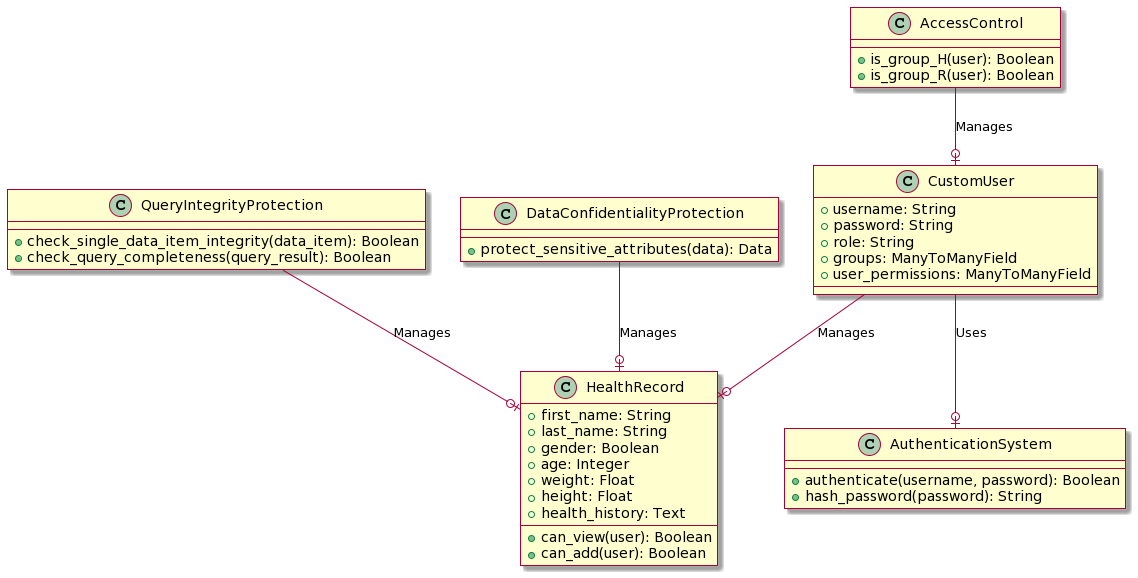
**3.4 Basic Data Confidentiality Protection**

*Implementation Approach:* To ensure the confidentiality of sensitive attributes, namely gender and age, the system incorporated a data protection mechanism. While simplifying the task, the system focused on preventing unauthorized access to these attributes, especially in scenarios where the cloud or local database management system might be queried. The protection mechanism was designed to safeguard against leaks of statistical information, ensuring that the percentage of data items with the same gender remains undisclosed.

**Project Report**

4.1 **System Architecture**

Our secure database-as-a-service system follows a modular architecture to ensure efficient handling of user authentication, access control, and data protection. The diagram below illustrates the key components and their interactions.



4.2 **Security Feature Implementation**

4.2.1 *User Authentication*

Our custom user authentication system successfully employs the username/password approach. Storing securely hashed passwords in the cloud enhances the overall security posture, mitigating the risk associated with password compromises.

4.2.2 *Basic Access Control Mechanism*

The implementation of a role-based access control mechanism ensures that Group H and Group R users have distinct permissions. Users from both groups can query data, with the returned results adhering to the defined access restrictions.

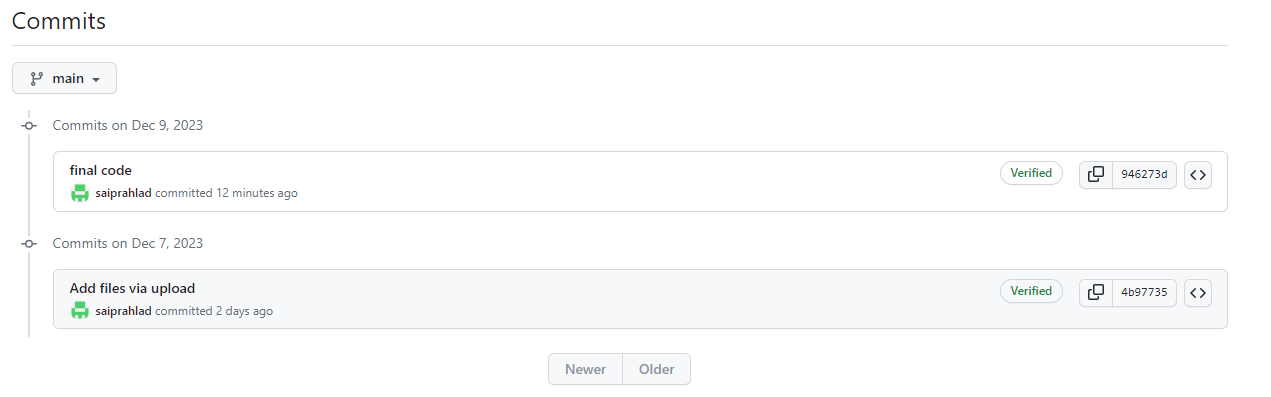
4.2.3 *Basic Query Integrity Protection*

The system provides robust protection against data manipulation in query results. Users can reliably detect modifications or fake data in a single data item, enhancing the integrity of the overall database. Additionally, the probability-based approach for query completeness ensures users are alerted to potential data removal.

4.2.4 *Basic Data Confidentiality Protection*

Sensitive attributes, such as gender and age, are effectively protected from unauthorized access. The system guards against leaks of statistical information, providing a secure environment for handling confidential data.

4.3 **Team Member Contributions and Github History**

**

4.4 **Limitations of the Project**

While our secure database-as-a-service system incorporates robust security features, certain limitations should be acknowledged:

* *Scalability:* The current system's scalability might be challenged in a real-world cloud environment with a massive user base. Further optimizations and considerations are necessary for scalability.
* *Real-time Monitoring:* The system lacks real-time monitoring capabilities, making it challenging to detect and respond promptly to security incidents. Integrating real-time monitoring features can enhance the system's responsiveness.
* *Cloud Provider Dependencies:* The system assumes a semi-trusted cloud environment. In a real-world scenario, dependence on specific cloud providers might introduce additional security considerations that need to be addressed.

Appendix

