**Data Security And Privacy**

Project - Team 22

Sarayu Papineni (811287764)

Rashvika Ponduri (811287449)

Chandra Vamsi Mummana (811288517)

[1. Introduction](#_Toc153125145)

[2. Basic Setup](#_Toc153125146)

[2.1 SQL Database System Setup](#_Toc153125147)

[2.2 Table Creation and Data Population](#_Toc153125148)

[3. Security Features](#_Toc153125149)

[3.1 User Authentication](#_Toc153125150)

[3.1.1 Implementation](#_Toc153125151)

[3.1.2 Reasoning](#_Toc153125152)

[3.2 Basic Access Control Mechanism](#_Toc153125153)

[3.2.1 Implementation](#_Toc153125154)

[3.2.2 Reasoning](#_Toc153125155)

[3.3 Basic Query Integrity Protection](#_Toc153125156)

[3.3.1 Single Data Item Integrity](#_Toc153125157)

[3.3.1.1 Implementation](#_Toc153125158)

[3.3.1.2 Reasoning](#_Toc153125159)

[3.3.2 Query Completeness](#_Toc153125160)

[3.3.2.1 Implementation](#_Toc153125161)

[3.3.2.2 Reasoning](#_Toc153125162)

[3.4 Basic Data Confidentiality Protection](#_Toc153125163)

[3.4.1 Implementation](#_Toc153125164)

[3.4.2 Reasoning](#_Toc153125165)

[4. Implementation](#_Toc153125166)

[4.1 System Design](#_Toc153125167)

[4.2 Implementation Details](#_Toc153125168)

[4.2.1 User Authentication](#_Toc153125169)

[4.2.1.1 Implementation](#_Toc153125170)

[4.2.1.2 Reasoning](#_Toc153125171)

[4.2.2 Basic Access Control Mechanism](#_Toc153125172)

[4.2.2.1 Implementation](#_Toc153125173)

[4.2.2.2 Reasoning](#_Toc153125174)

[4.2.3 Basic Query Integrity Protection](#_Toc153125175)

[4.2.3.1 Implementation](#_Toc153125176)

[4.2.3.2 Reasoning](#_Toc153125177)

[4.2.4 Basic Data Confidentiality Protection](#_Toc153125178)

[4.2.4.1 Implementation](#_Toc153125179)

[4.2.4.2 Reasoning](#_Toc153125180)

[4.3 Team Contributions](#_Toc153125181)

[4.4 Limitations](#_Toc153125182)

[4.4.1 Scalability](#_Toc153125183)

[4.4.2 External Dependencies](#_Toc153125184)

[4.4.3 Real-world Cloud Deployment](#_Toc153125185)

[4.4.4 Limited Attribute Protection](#_Toc153125186)

[5. Conclusion](#_Toc153125187)

# 1. Introduction

The need for scalable and dependable records storage has made Database-as-a-Service (DBaaS) essential in the technology of expanding cloud app usage. While cloud-primarily based databases are smooth to use, they come with security issues, with unauthorized access to personal facts being a excellent threat. This project intends to develop, implement, and evaluate a secure DBaaS system, focusing on healthcare information. We employ Java and MySQL, emulating a cloud environment while running the system locally.

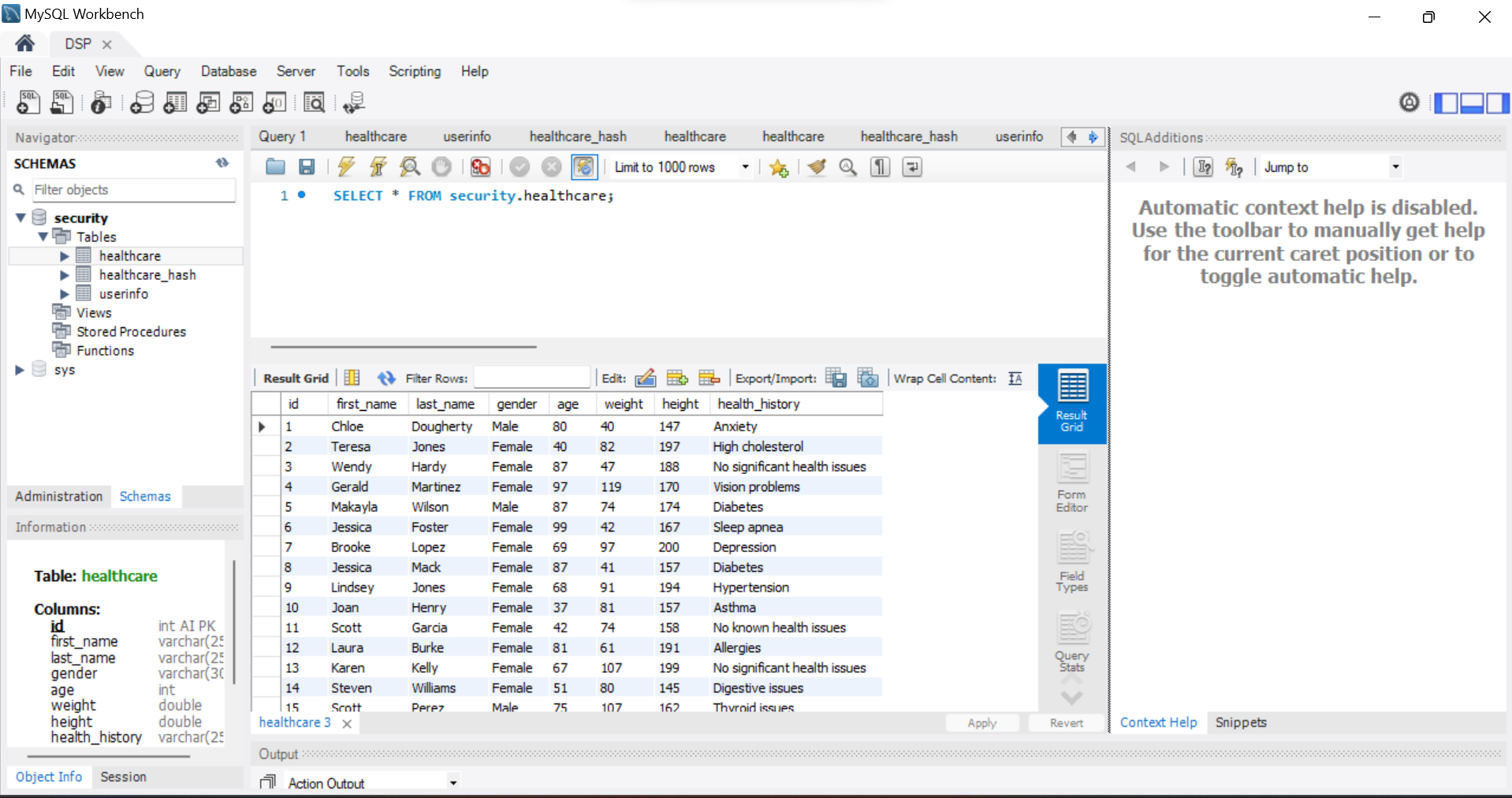
# 2. Basic Setup

## 2.1 SQL Database System Setup

For our project, we selected MySQL as the relational database management system (RDBMS). While our implementation is implemented locally, it matches the structure and features of a cloud-based MySQL environment. We chose MySQL because of its extensive use, powerful features, and interoperability with Java.

## 2.2 Table Creation and Data Population

The healthcare information table is designed to mirror real-world settings, containing vital variables such as first name, last name, gender, age, weight, height, and health history. To confirm the system's robustness, we populated the database with a minimum of 100 data items, encompassing varied values for a full test environment.



# 3. Security Features

Security is a vital aspect of our DBaaS solution. We've incorporated numerous features to address authentication, access control, query integrity protection, and data confidentiality.

## 3.1 User Authentication

### 3.1.1 Implementation

Our system implements a bespoke user authentication mechanism, requiring a valid username/password combination for access. During user registration, passwords are hashed using the SHA-256 technique before being saved in the database.

### 3.1.2 Reasoning

Custom authentication boosts security by ensuring that the system doesn't rely exclusively on the database's built-in authentication processes. Storing hashed passwords prevents unauthorized access, even in the event of a data breach.

## 3.2 Basic Access Control Mechanism

### 3.2.1 Implementation

We created the user groups 'H' and 'R.' Individuals belonging to group 'H' possess complete access to every field, which includes the capability to create new data items. First and last name fields are not accessible to Group 'R' users, who have restricted access. Based on the user's group, query results are dynamically filtered.

### 3.2.2 Reasoning

Users can only interact with the data they are authorized to see or edit thanks to this access control system. The solution ensures data confidentiality and stops accidental data exposure by limiting access at the query level.

## 3.3 Basic Query Integrity Protection

### 3.3.1 Single Data Item Integrity

### 3.3.1.1 Implementation

In order to create a strong integrity protection method, we used hash functions to create checksums for every piece of data. The system recalculates checksums and compares them with the stored values while the query is being executed.

### 3.3.1.2 Reasoning

With this method, the system can identify any manipulation or alteration of specific data elements. Users are able to confirm the accuracy of the data they have retrieved from the database, irrespective of their affiliation.

### 3.3.2 Query Completeness

### 3.3.2.1 Implementation

Our system presents a probabilistic method for determining whether query results are full. The system is able to determine which data items in the query result are missing by keeping statistical information about the dataset.

### 3.3.2.2 Reasoning

An extra degree of data integrity is offered by probabilistic completeness checks. This method improves the system's capacity to spot any data omissions; however, it is not infallible.

## 3.4 Basic Data Confidentiality Protection

### 3.4.1 Implementation

Gender and age are examples of sensitive attributes that are identified and shielded from unwanted access. The system makes sure that protected properties cannot be retrieved by queries that are started by the local database management system or the cloud.

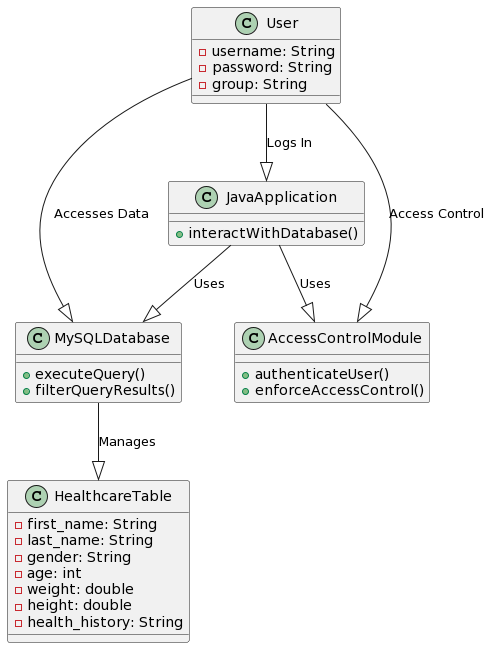
### 3.4.2 Reasoning

Sensitive data is protected by this security feature from illegal access or possible leaks. The solution reduces the possibility of sensitive healthcare data exposure by specifically blocking access to specific properties.

# 4. Implementation

## 4.1 System Design

The architecture of our DBaaS system is illustrated in Figure 1. The system comprises three main components: the Java application, the MySQL database, and the access control module.



**Figure 1: System Architecture**

Users can interact with the database more easily because the Java application serves as their interface. According to the user's group, the access control module dynamically filters query results, making sure that the access levels are followed.

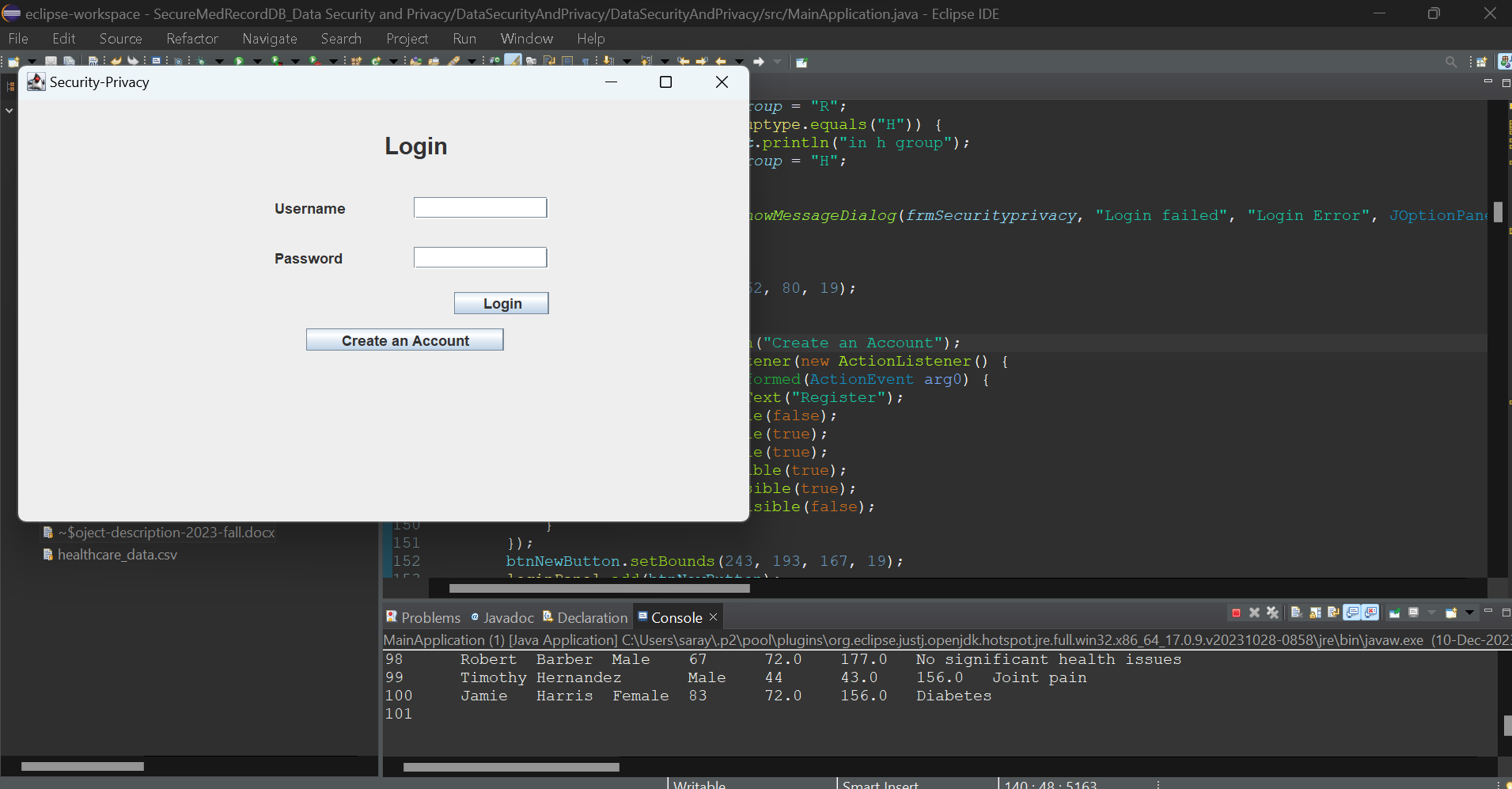
## 4.2 Implementation Details

### 4.2.1 User Authentication

### 4.2.1.1 Implementation

Our custom authentication system involves the following steps:

* User registration: The system uses SHA-256 method to hash the user password.
* User Login: The system hashes the password entered by the user and compares it to the hash stored when the user tries to login.



### 4.2.1.2 Reasoning

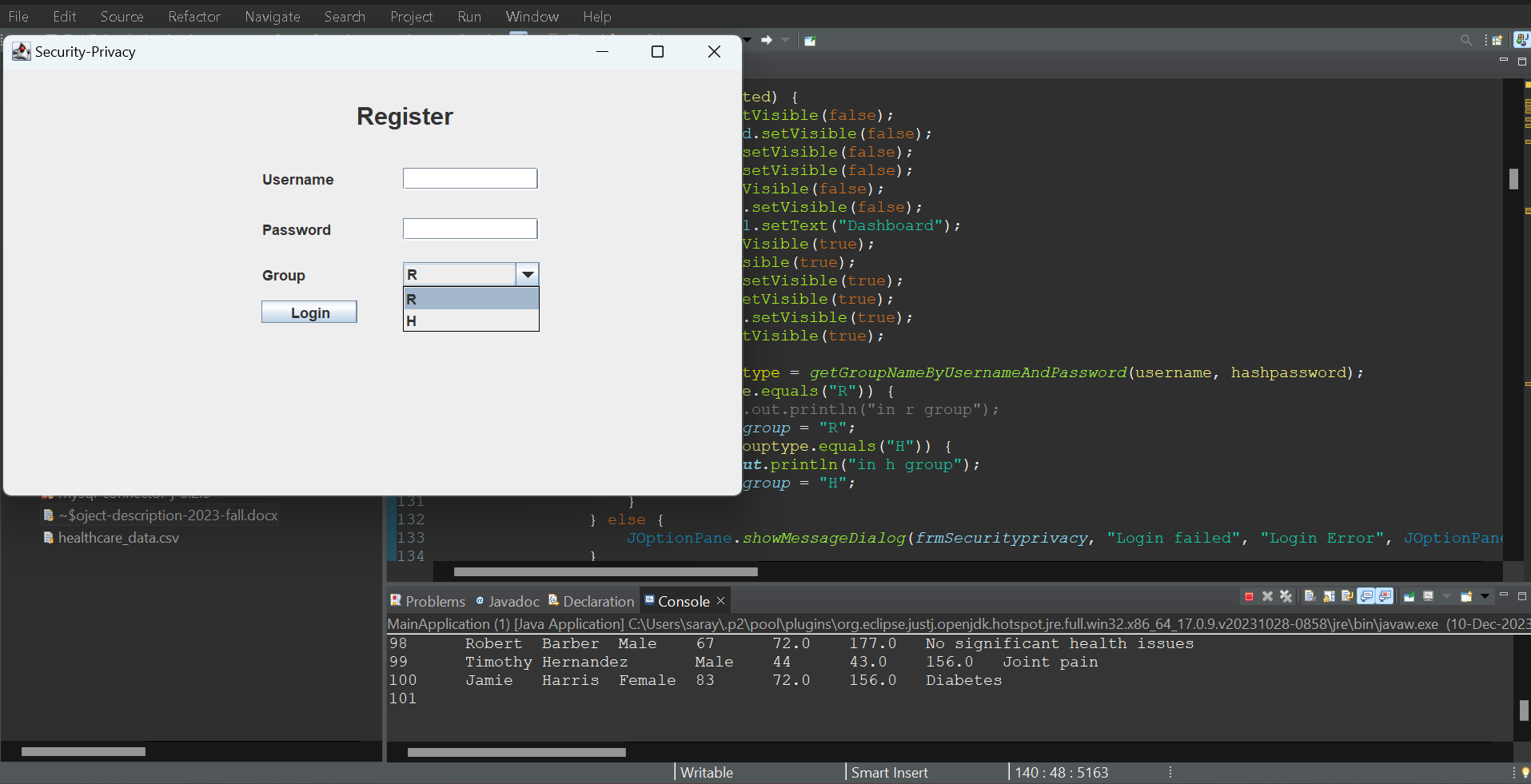
Personalized authentication enables customized security protocols. By keeping plain-text passwords hidden, hashing improves security even in the event that a database hack occurs.

### 4.2.2 Basic Access Control Mechanism

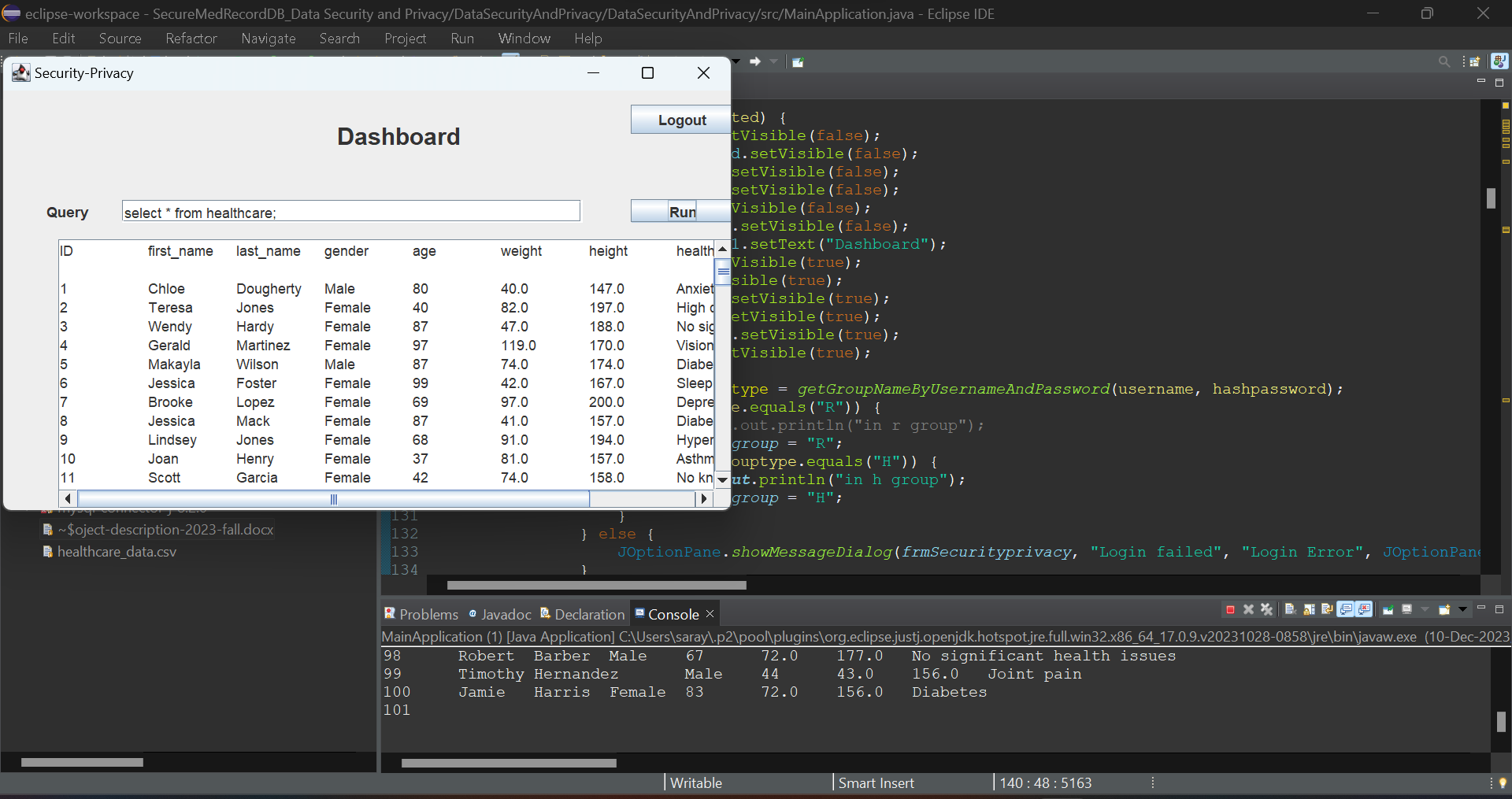
### 4.2.2.1 Implementation

Access control is enforced through user groups and the dynamic filtering of query results. The system follows these steps:

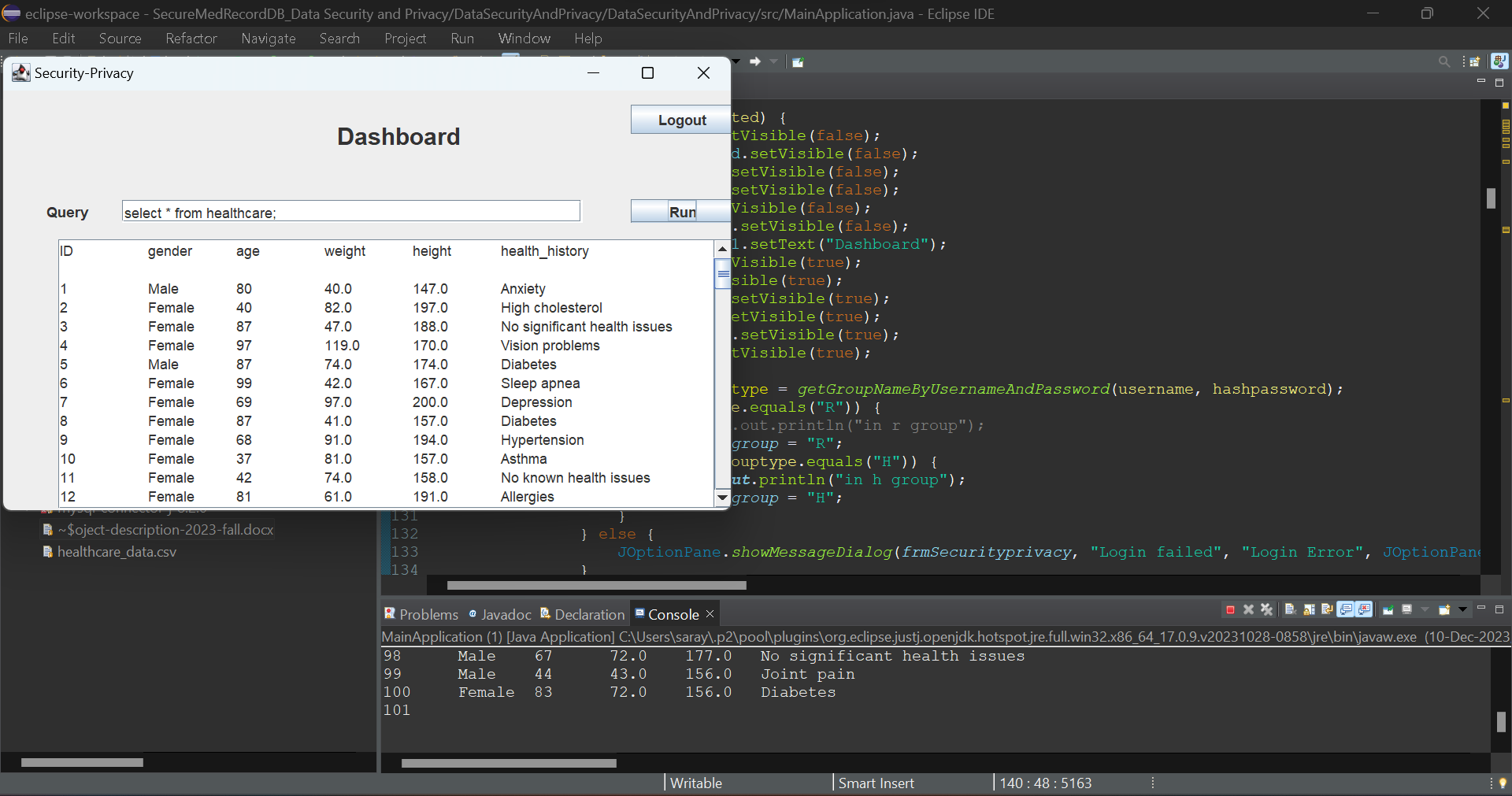
* User grouping: Users are assigned to either group 'H' or 'R' during registration.
* Query execution: The system identifies the user's group and dynamically adjusts the query to exclude restricted fields for group 'R' users.



**The below screenshot belongs to H – Group access control**

****

**The below screenshot belongs to R – Group access control**

****

### 4.2.2.2 Reasoning

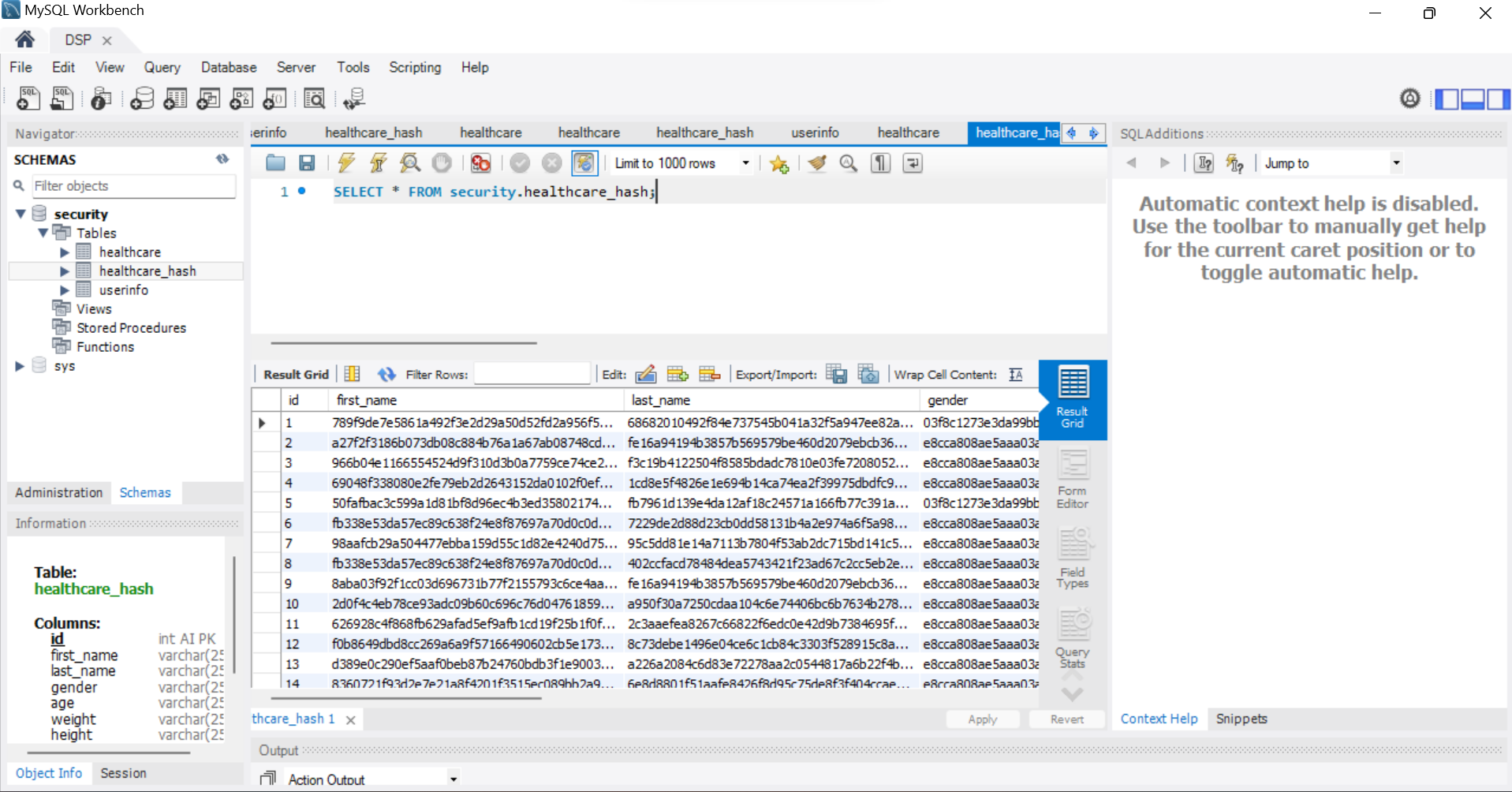
This method ensures that every user interacts with the data in accordance with their allocated rights and offers fine-grained control over user access. Access control enforcement is made simpler by the dynamic filtering approach.

### 4.2.3 Basic Query Integrity Protection

### 4.2.3.1 Implementation

Integrity protection is achieved through the use of hash functions. The system follows these steps:

* Checksum generation: For each data item, the system calculates a checksum based on its attributes.
* Query execution: During query execution, the system recalculates checksums and verifies their consistency with stored values.



### 4.2.3.2 Reasoning

Checksums are used to guarantee that users can confirm the accuracy of particular data items. Hash functions are selected based on how well they perform data integrity checks and how strong their cryptography is.

### 4.2.4 Basic Data Confidentiality Protection

### 4.2.4.1 Implementation

Protection of sensitive attributes involves explicit restrictions in query execution. The system follows these steps:

* Identification of sensitive attributes: The system identifies attributes such as gender and age as sensitive.
* Query execution: The system prevents the retrieval of sensitive attributes in query results.

### 4.2.4.2 Reasoning

This technique satisfies the project's basic data confidentiality protection criterion by preventing unauthorized access to sensitive information. The technology proactively reduces the danger of data leakage by identifying specific features as sensitive.

## 4.3 Team Contributions

## The group worked well together to create a safe Database-as-a-Service (DBaaS) system designed specifically for the handling of medical data. With her background in security and databases, Rashvika took the lead in selecting MySQL, setting up the SQL database, and putting user authentication and access control in place. Sarayu introduced features for query integrity protection, probabilistic completeness checks, data confidentiality, and the final report. Sarayu possesses competence in both query integrity and data secrecy. Specializing in both Java application and system design, Chandra Vamsi oversaw the development of the Java applications as well as the access control module. We successfully worked together as a team to complete the job. Each participant made a substantial contribution to a variety of implementation-related areas.

## 4.4 Limitations

There are restrictions to take into account, especially when it comes to security and privacy, even though our system handles a number of security concerns:

### 4.4.1 Scalability

When working with a substantially larger dataset, the existing solution can encounter scaling issues. Because the system uses probabilistic techniques to verify the correctness of queries, scaling to large databases may have an effect on performance.

### 4.4.2 External Dependencies

A possible weakness in the system is the hash functions' dependency on third-party libraries. To lessen this reliance, future versions should investigate the incorporation of unique cryptographic functions.

### 4.4.3 Real-world Cloud Deployment

Even if we only create a local simulation of a cloud environment, actual cloud platform deployment may present new difficulties. Consideration must be given to variables including network latency, regional dispersion, and different user access patterns.

### 4.4.4 Limited Attribute Protection

The protection of particular qualities designated as sensitive is the main goal of the current implementation. Future improvements might include a more flexible approach to attribute sensitivity, enabling users to mark some attributes as sensitive in accordance with situational demands.

# 5. Conclusion

To sum up, our DBaaS solution shows how well security protections are included to safeguard patient information. A secure environment is facilitated by the combination of the custom authentication system, access control mechanism, query integrity protection, and data confidentiality protection. Even with the system's shortcomings—most notably in terms of attribute preservation and scalability—it offers a solid framework for future advancements. The commit history, which details our cooperative team effort, demonstrates how well we worked together to accomplish the project's objectives..

**Github Link**

<https://github.com/Sarayup3704/SecureMedRecordDB>