**ABSTRACT**

The Scholarship and Mentorship Management System is an encompassing platform engineered to streamline and elevate educational processes by efficient handling of scholarship programs and mentorship engagements. This project endeavors to establish a robust and user-centric system that addresses the intricacies of academic support while ensuring data accuracy, security, and accessibility.

The system incorporates various modules, including scholarship application management, mentor-student pairing, academic record tracking, and financial disbursement. A resilient and scalable database forms the core foundation of the application, facilitating seamless storage, retrieval, and manipulation of educational data. The selection of a dependable database platform is pivotal to ensure optimal performance and adaptability to the evolving landscape of educational needs.

The efficacy of the Scholarship and Mentorship Management System hinges on the adept integration of database management principles to ensure data precision, dependability, and availability. Through leveraging a potent and scalable database solution, the system aims to streamline educational support processes, alleviate administrative burdens, and ultimately augment academic outcomes for students and mentors alike.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Chapter No.** | **Title** | **Page No.** |
|  | **INTRODUCTION** | **10** |
|  | **PROBLEM DEFINITION** | **11** |
|  | **ER MODEL** | **12** |
|  | **ER TO RELATIONAL MAPPING** | **13** |
|  | **DDL STATEMENTS** | **15** |
|  | **DML STATEMENTS** | **20** |
|  | **QUERIES (SIMPLE QUERY AND UPDATE AND DELETE OPERATION, CORRELATED QUERY AND NESTED QUERY)** | **23** |
|  | **STORED PROCEDURE, FUNCTIONS AND TRIGGERS** | **30** |
|  | **FRONT END DEVELOPMENT** | **31** |
| **REFERENCES/BIBLIOGRAPHY** | | **35** |
| **APPENDIX A DEFINITIONS, ACRONYMS AND ABBREVIATIONS** | | **36** |
|  | |  |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **Table No.** | **Title** | **Page No.** |
| **1.1** | **Student table** | **14** |
| **1.2** | **Scholarship table** | **14** |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Title** | **Page No.** |
| **1.0** | **ER Diagram** | **11** |
| **1.1** | **Relational Mapping** | **14** |
| **1.2** | **Login Page** | **29** |
| **1.3** | **Dashboard** | **29** |
| **1.4** | **View page** | **30** |
| **1.5** | **Adding data through frontend** | **31** |

**1.6 Deleting student data through frontend 32**

**1. INTRODUCTION**

In today’s dynamic educational landscape, managing scholarship programs and fostering meaningful mentorship engagements are pivotal for student success and academic advancement. The Scholarship and Mentorship Management System is an innovative solution tailored to optimize these critical components of educational support.

This project aims to introduce a comprehensive platform that harmonizes scholarship application processes, facilitates efficient mentor-student pairings, and provides robust academic record tracking. By leveraging the power of modern database management systems, this system endeavours to simplify administrative tasks, enhance data integrity, and empower educational institutions to offer tailored support to students and mentors.

The dynamic nature of educational support requires a seamless and scalable database infrastructure to manage diverse datasets effectively. This system prioritizes the integration of reliable and adaptable database solutions to ensure the fluidity of scholarship disbursements, mentor allocations, and academic progress tracking.

Through this project, we seek to elevate the educational experience by offering a centralized, user-friendly interface that fosters effective communication, strategic allocation of resources, and data-driven decision-making. The Scholarship and Mentorship Management System strives to be the catalyst for improving educational outcomes and nurturing a supportive ecosystem for students and mentors.

**2. PROBLEM DEFINITION**

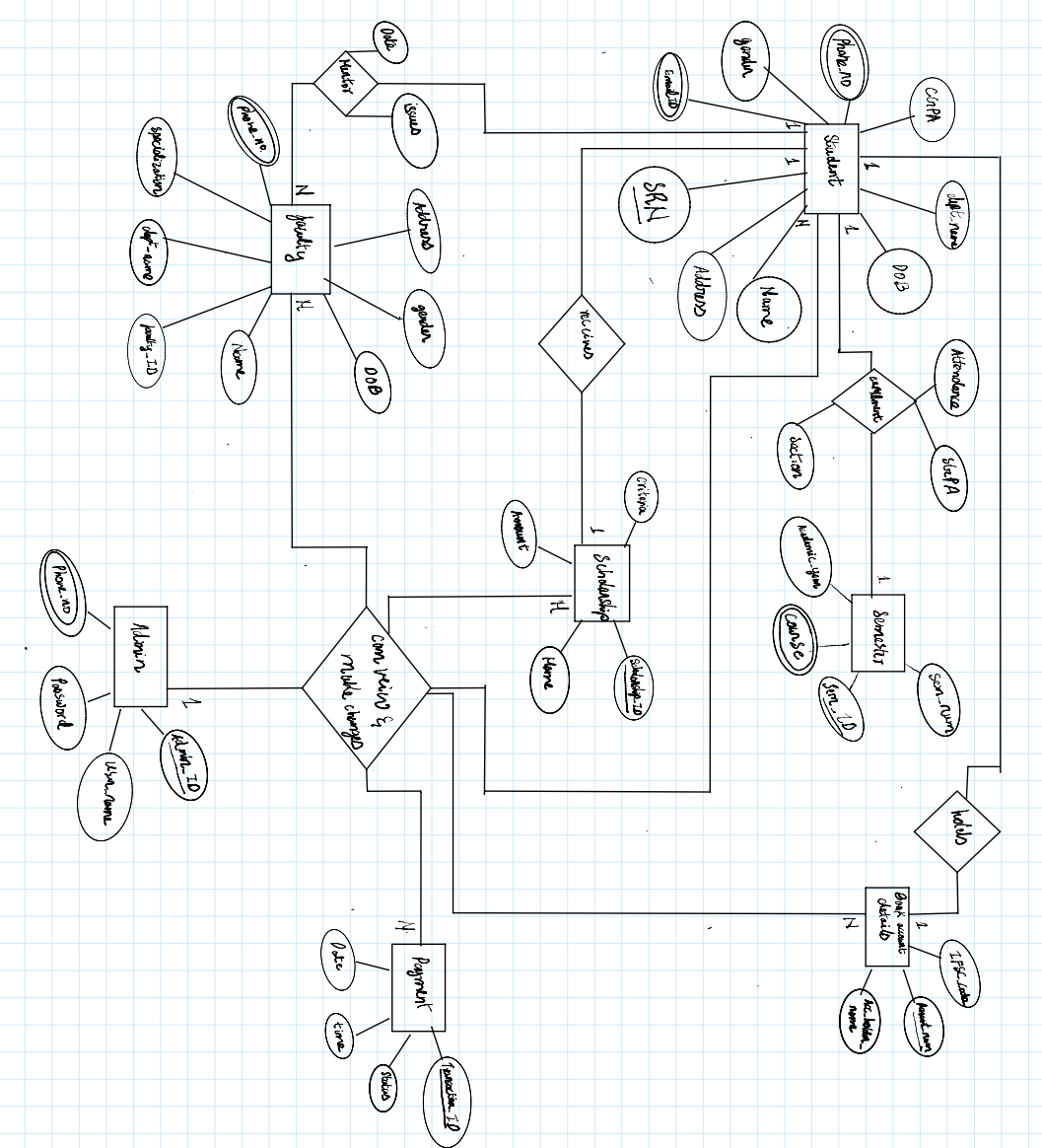
In the realm of education, the process of managing scholarship programs and mentorship engagements faces multifaceted challenges that hinder streamlined operations and optimal outcomes. Current systems often lack integration, resulting in fragmented data management, inefficient application processes, and inadequate support for mentor-student interactions. These challenges contribute to administrative complexities, hindered accessibility to educational resources, and the inability to harness data insights for informed decision-making.

This project addresses these prevalent issues by aiming to develop a comprehensive Scholarship and Mentorship Management System that streamlines the application, evaluation, and disbursement processes for scholarships, as well as facilitates efficient mentor-student pairings. The system endeavors to overcome the existing limitations by integrating robust database management principles, ensuring data accuracy, security, and accessibility while offering a user-friendly interface for stakeholders.

Key challenges to be tackled include the lack of centralized data management leading to disparate information silos, inefficient scholarship application procedures causing delays and confusion, and a dearth of streamlined platforms for mentor-student interactions. Additionally, the project aims to navigate complexities related to data security, scalability, and the adaptability of the system to evolving educational needs.

By addressing these challenges, the aim is to create a solution that harmonizes scholarship and mentorship management, empowering educational institutions to offer personalized support, enhance academic success, and foster a conducive environment for both students and mentors to thrive.

**3. ER MODEL**

****

**4. ER TO RELATIONAL MAPPING**

**4.1 STEPS OF ALGORITHM FOR CHOOSEN PROBLEM**

**1. Identify Entities:**

List all the entities in the ER diagram. In our case, entities include Student, Semester, Bank account details, Scholarship, Payment, Admin and Faculty.

2. Identify Relationships:

Identify the relationships between the entities. Look for associations and dependencies. In our case, relationships includes the association between Student and Semester (via SRN and semid),Student and Bankaccount details(via SRN and account number ), Student and Scholarship (via SRN ), Scholarship and Payment (via scholarship\_id) ,

Admin to every other entities and Student and Faculty (via SRN ans Faculty\_id)

3. Identify Attributes:

For each entity, list the attributes. In our case, Student attributes could include Name, SRN, Semester, Gender, CGPA, email, address, DOB, phone number and department. Similarly, other entities have their specific attributes.

4. Identify Primary Keys:

Determine the primary key for each table. In our case, SRN could be the primary key for the Student table, scholarship\_id for the Scholarship table, a combination of Semester ID and SRN for the Enrols table, sem\_id for the Semester table, comnination of SRN ,fam date and faculty\_id for the Mentor table,transaction\_id for Payments table , admin\_id for admin table and faculty\_id for the Faculty table.

Identify Foreign Keys:

5. Identify foreign keys that establish relationships between tables.

In our case, SRN and sem\_id in theenrols table would be a foreign key referencing the SRN and semester id in the Student table and semester table. Scholarship\_id in Paymnets table referencing the Scholarshipid in Scholarship table. Similarly, other foreign key relationships can be identified.

6. Define Data Types:

Determine the data types for each attribute. Common data types include INT, VARCHAR, DATE, etc. For example, Name could be VARCHAR, SRN could be VARCHAR and so on.

7. Draw the Tables:

Create a table for each entity based on the identified attributes, primary keys, and foreign keys. Make sure to include the appropriate data types. For example:

Student Table:

| SRN (PK) | Name | Address| DOB| Dept name| Gender | CGPA | Phone no(Unique)| Email (Unique) |

|------------|-------|---------|------|------------|---------|--------|--------------------|-----------------|

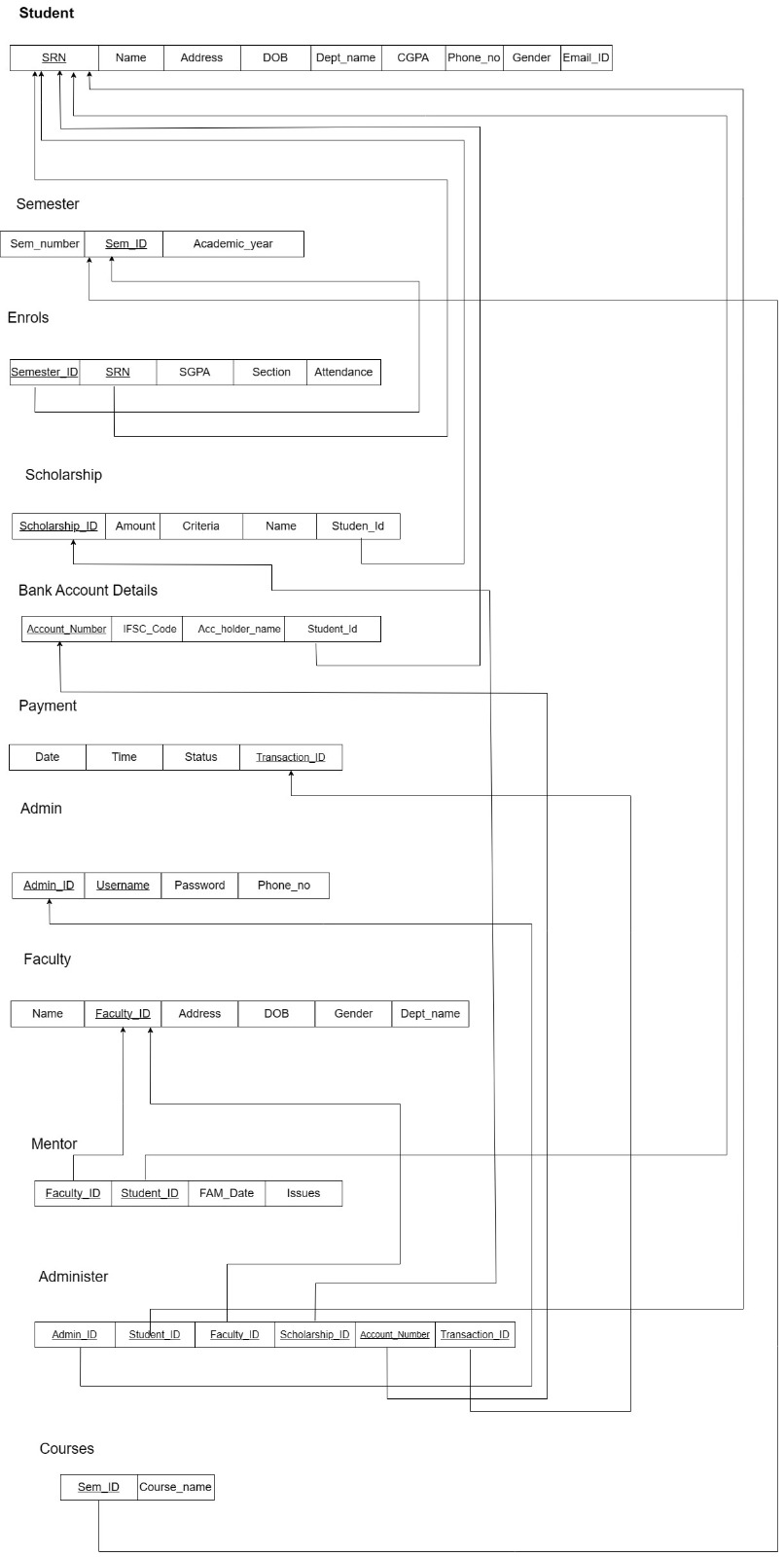
Scholarship Table:

| Scholarship\_id(PK) | amount |criteria | name | student\_id(FK) |

|--------------------------|----------|---------|--------|---------------------|

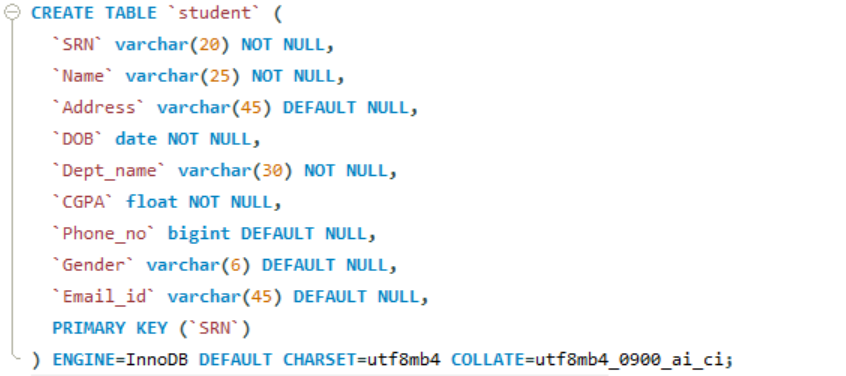
Similary this process for the remaining tables.

**4.2 COMPLETE DIAGRAM OF RELATIONAL MAPPING**

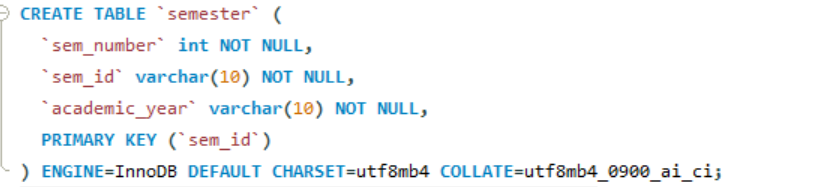
****

**5. DDL STATEMENTS**

CREATE TABLE student



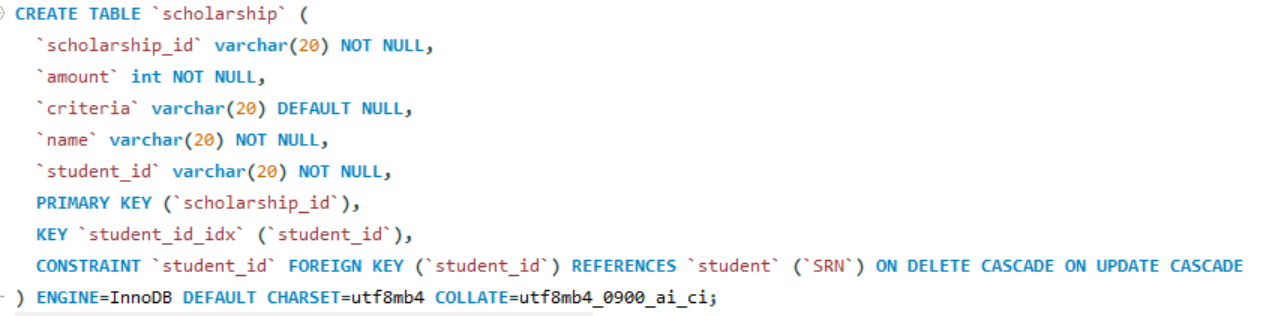
CREATE TABLE semester



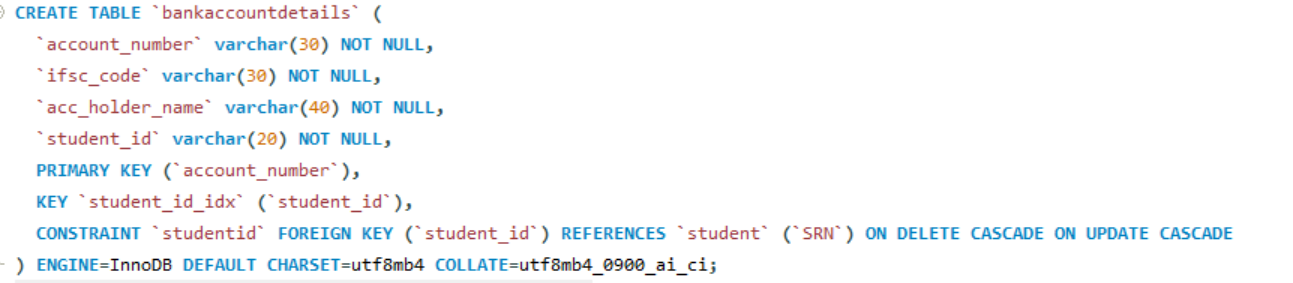
CREATE TABLE enrols



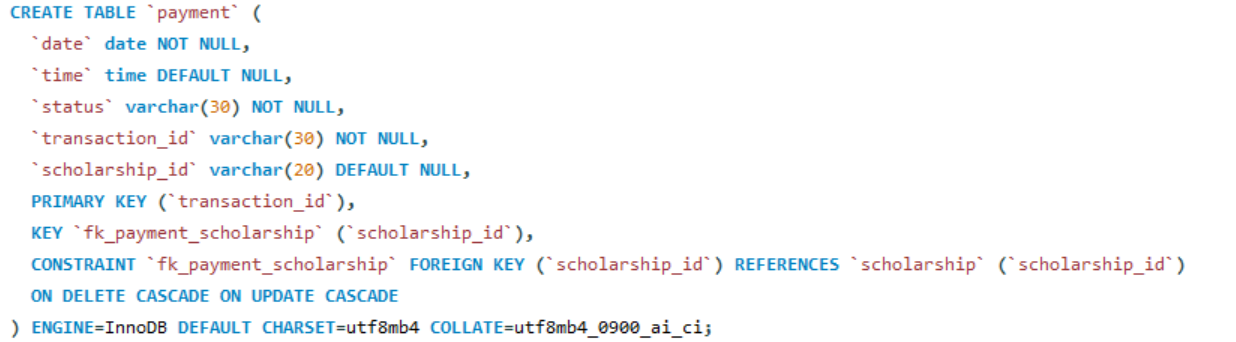
CREATE TABLE scholarship



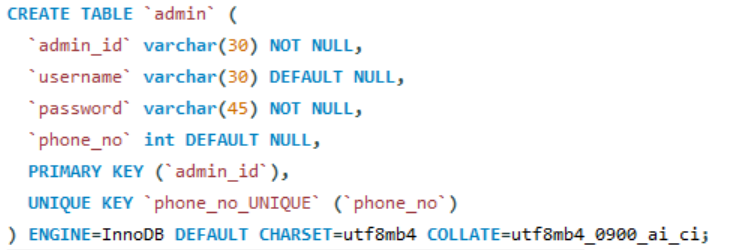
CREATE TABLE bankaccountdetails



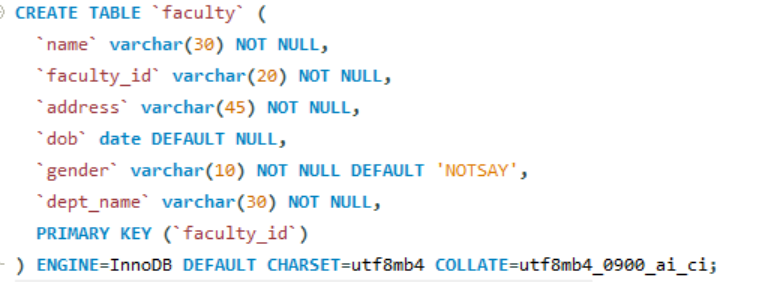
CREATE TABLE payment



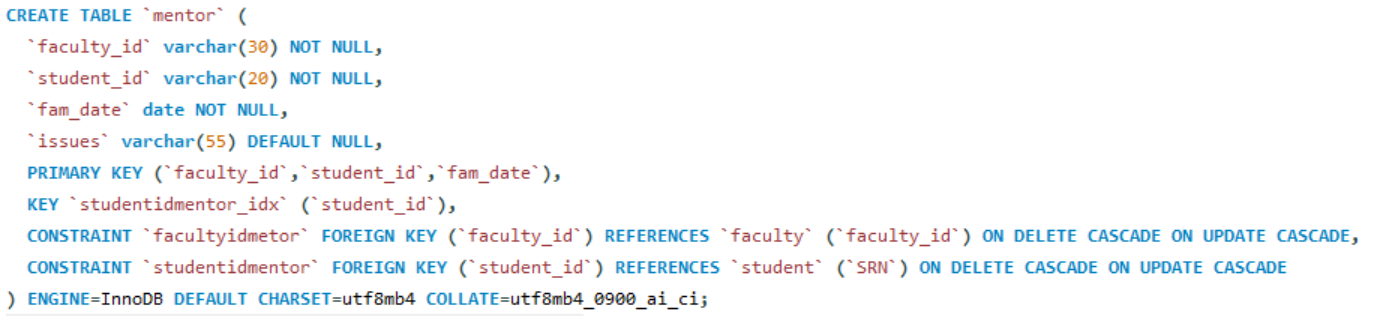
CREATE TABLE admin



CREATE TABLE faculty



CREATE TABLE mentor



CREATE TABLE administer



# 6. DML STATEMENTS

Update admin table:

def update\_admin():

    result = view\_all\_data\_admin()

    df = pd.DataFrame(result, columns=['Admin ID', 'Admin Name', 'Admin Password', 'Admin Phone Number'])

    with st.expander("Current Admins"):

        st.dataframe(df)

    list\_of\_admins = [i[0] for i in view\_only\_admin\_ids()]

    selected\_admin = st.selectbox("Admin to Edit", list\_of\_admins)

    selected\_result = get\_admin(selected\_admin)

    if selected\_result:

        admin\_id = selected\_result[0][0]

        admin\_name = selected\_result[0][1]

        admin\_password = selected\_result[0][2]

        admin\_phone\_no = selected\_result[0][3]  # Assuming phone\_no is at index 3, modify if needed

        # Layout for updating admin data

        col1, col2, col3 = st.columns(3)

        with col1:

            new\_admin\_id = st.text\_input("ID:", admin\_id)

            new\_admin\_name = st.text\_input("Name:", admin\_name)

        with col2:

            new\_admin\_password = st.text\_input("Password:", admin\_password)

        with col3:

            new\_admin\_phone\_no = st.text\_input("Phone Number:", admin\_phone\_no)

        if st.button("Update Admin"):

            # Pass all updated fields to the edit\_admin\_data function

            edit\_admin\_data(new\_admin\_id, new\_admin\_name, new\_admin\_password, new\_admin\_phone\_no, admin\_id)

            st.success("Successfully updated Admin: {} to {}".format(admin\_name, new\_admin\_name))

    result2 = view\_all\_data\_admin()

    df2 = pd.DataFrame(result2, columns=['Admin ID', 'Admin Name', 'Admin Password', 'Admin Phone Number'])

    with st.expander("Updated Admin Data"):

        st.dataframe(df2)

Delete bank details:

def delete\_bankaccountdetails():

    result = view\_all\_data\_bankaccountdetails()

    df = pd.DataFrame(result, columns=['Account Number', 'IFSC Code', 'Account Holder Name', 'Student ID'])

    with st.expander("Current Bank Account Details"):

        st.dataframe(df)

    list\_of\_accounts = [i[0] for i in view\_only\_bankaccountdetails\_ids()]

    selected\_account = st.selectbox("Task to Delete", list\_of\_accounts)

    st.warning("Do you want to delete Bank Account ::{}".format(selected\_account))

    if st.button("Delete Bank Account"):

        delete\_bankaccountdetails\_data(selected\_account)

        st.success("Bank Account details have been deleted successfully")

    new\_result = view\_all\_data\_bankaccountdetails()

    df2 = pd.DataFrame(new\_result, columns=['Account Number', 'IFSC Code', 'Account Holder Name', 'Student ID'])

    with st.expander("Updated Bank Account Details"):

        st.dataframe(df2)

Delete Faculty:

def delete\_faculty():

    result = view\_all\_data\_faculty()

    df = pd.DataFrame(result, columns=['Name', 'Faculty ID', 'Address', 'DOB', 'Gender', 'Department Name'])

    with st.expander("Current Faculty Data"):

        st.dataframe(df)

    list\_of\_faculty = [i[0] for i in view\_only\_faculty\_ids()]

    selected\_faculty = st.selectbox("Task to Delete", list\_of\_faculty)

    st.warning("Do you want to delete Faculty ::{}".format(selected\_faculty))

    if st.button("Delete Faculty"):

        delete\_faculty\_data(selected\_faculty)

        st.success("Faculty has been deleted successfully")

    new\_result = view\_all\_data\_faculty()

    df2 = pd.DataFrame(new\_result, columns=['Name', 'Faculty ID', 'Address', 'DOB', 'Gender', 'Department Name'])

    with st.expander("Updated Faculty Data"):

        st.dataframe(df2)

Reading payment details:

def read\_payment():

    result = view\_all\_data\_payment()

    df = pd.DataFrame(result, columns=['Payment ID', 'Amount', 'Date', 'Student ID'])

    with st.expander("View all Payments"):

        st.dataframe(df)

Reading semester details:

def read\_semester():

    result = view\_all\_data\_semester()

    df = pd.DataFrame(result, columns=['Semester Number', 'Semester ID', 'Academic Year'])

    with st.expander("View all Semesters"):

        st.dataframe(df)

Adding student details:

def add\_data\_student(SRN, Name, Address, DOB, Dept\_name, CGPA, Phone\_no, Gender, Email\_id):

    c.execute('INSERT INTO student(SRN, Name, Address, DOB, Dept\_name, CGPA, Phone\_no, Gender, Email\_id) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)',

              (SRN, Name, Address, DOB, Dept\_name, CGPA, Phone\_no, Gender, Email\_id))

    mydb.commit()

Adding scholarship details:

def add\_data\_scholarship(scholarship\_id, amount, criteria, name, student\_id):

    c.execute('INSERT INTO scholarship(scholarship\_id, amount, criteria, name, student\_id) VALUES (%s, %s, %s, %s, %s)',

              (scholarship\_id, amount, criteria, name, student\_id))

    mydb.commit()

**7. QUERIES**

**UPDATE OPERATION**

def update\_admin():

    result = view\_all\_data\_admin()

    df = pd.DataFrame(result, columns=['Admin ID', 'Admin Name', 'Admin Password', 'Admin Phone Number'])

    with st.expander("Current Admins"):

        st.dataframe(df)

    list\_of\_admins = [i[0] for i in view\_only\_admin\_ids()]

    selected\_admin = st.selectbox("Admin to Edit", list\_of\_admins)

    selected\_result = get\_admin(selected\_admin)

    if selected\_result:

        admin\_id = selected\_result[0][0]

        admin\_name = selected\_result[0][1]

        admin\_password = selected\_result[0][2]

        admin\_phone\_no = selected\_result[0][3]  # Assuming phone\_no is at index 3, modify if needed

        # Layout for updating admin data

        col1, col2, col3 = st.columns(3)

        with col1:

            new\_admin\_id = st.text\_input("ID:", admin\_id)

            new\_admin\_name = st.text\_input("Name:", admin\_name)

        with col2:

            new\_admin\_password = st.text\_input("Password:", admin\_password)

        with col3:

            new\_admin\_phone\_no = st.text\_input("Phone Number:", admin\_phone\_no)

        if st.button("Update Admin"):

            # Pass all updated fields to the edit\_admin\_data function

            edit\_admin\_data(new\_admin\_id, new\_admin\_name, new\_admin\_password, new\_admin\_phone\_no, admin\_id)

            st.success("Successfully updated Admin: {} to {}".format(admin\_name, new\_admin\_name))

    result2 = view\_all\_data\_admin()

    df2 = pd.DataFrame(result2, columns=['Admin ID', 'Admin Name', 'Admin Password', 'Admin Phone Number'])

    with st.expander("Updated Admin Data"):

        st.dataframe(df2)

**DELETE OPERATION**

def delete\_payment():

    result = view\_all\_data\_payment()

    df = pd.DataFrame(result, columns=['Date', 'Time', 'Status', 'Transaction ID','Scholarship ID'])

    with st.expander("Current Payment Data"):

        df['Time'] = df['Time'].astype(str)

        st.dataframe(df)

    list\_of\_payments = [i[0] for i in view\_only\_payment\_ids()]

    selected\_payment = st.selectbox("Task to Delete", list\_of\_payments)

    st.warning("Do you want to delete Payment ::{}".format(selected\_payment))

    if st.button("Delete Payment"):

        delete\_payment\_data(selected\_payment)

        st.success("Payment has been deleted successfully")

    new\_result = view\_all\_data\_payment()

    df2 = pd.DataFrame(new\_result, columns=['Date', 'Time', 'Status', 'Transaction ID','Scholarship ID'])

    with st.expander("Updated Payment Data"):

        df2['Time'] = df2['Time'].astype(str)

        st.dataframe(df2)

**7.4 CORRELATED QUERY**

def greater\_avg\_sgpa():

    table\_name = 'enrols'

    # Build the SQL query

    sql\_query = f"SELECT SRN AS Student\_id, SGPA AS Students\_with\_SGPA\_More\_than\_avg\_SGPA " \

                f"FROM {table\_name} e " \

                f"WHERE SGPA > (SELECT AVG(e2.SGPA) FROM {table\_name} e2 WHERE e2.sem\_id = e.sem\_id)"

**7.5 NESTED QUERY**

def greater\_avg\_cgpa():

    table\_name = 'student'

    # Build the SQL query

    sql\_query = f"SELECT CGPA AS Students\_with\_CGPA\_More\_than\_avg\_CGPA, SRN AS Student\_id , Name as Name  " \

                f"FROM {table\_name} " \

                f"WHERE CGPA > (SELECT AVG(CGPA) FROM {table\_name})"

    # Execute the query

    c.execute(sql\_query)

    # Fetch the result

    result\_sum = c.fetchall()

    # Create a DataFrame to store the result

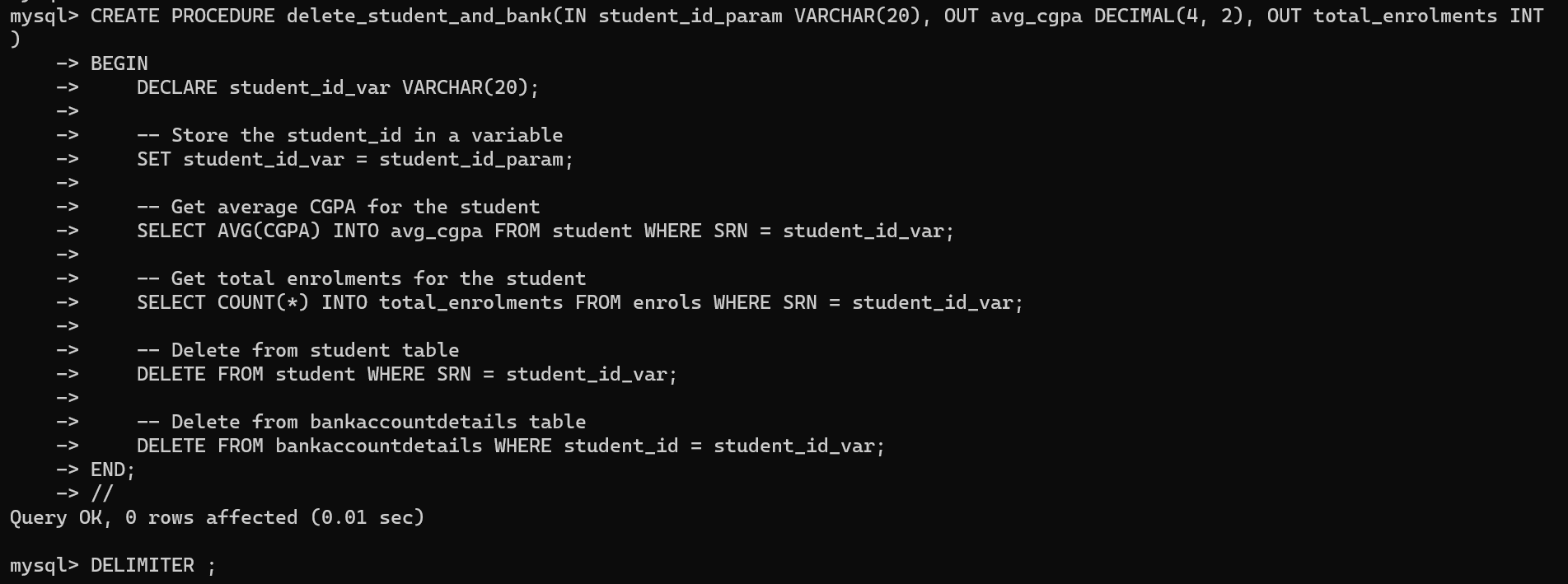
    df\_total\_scholarship = pd.DataFrame(result\_sum, columns=['Students\_with\_CGPA\_More\_than\_avg\_CGPA', 'Student\_id', 'Name'])

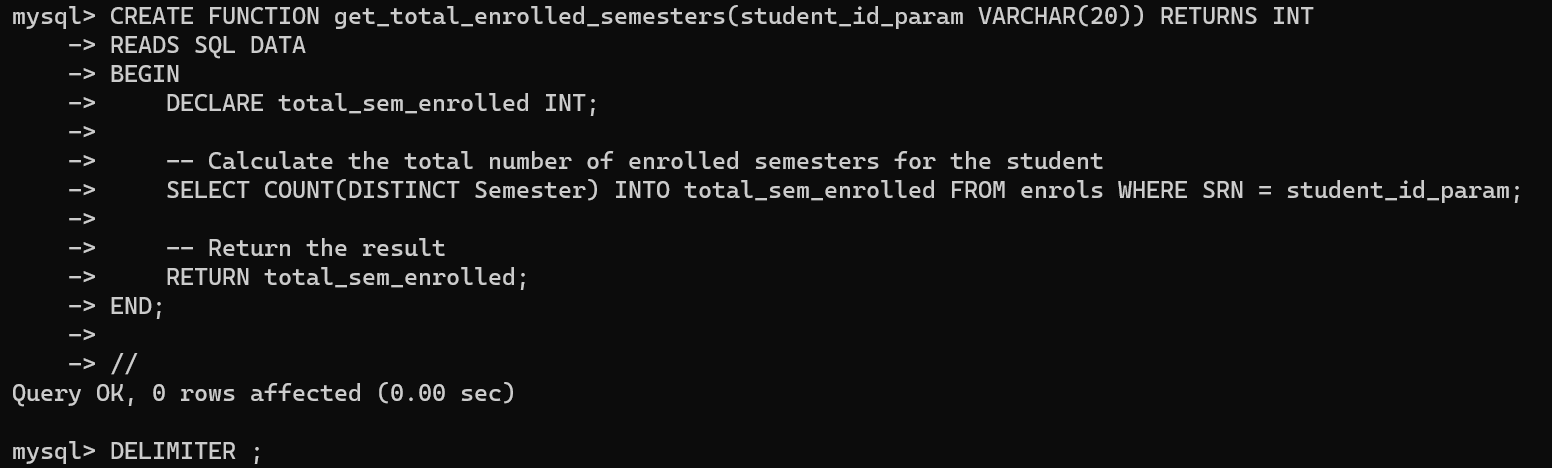
    # Display the result

    st.write(df\_total\_scholarship)

**8. STORED PROCEDURES, FUCNTIONS AND TRIGGERS**

**8.1 STORED PROCEDURES OR FUNCTIONS**

****

****

**8.2 TRIGGERS**

DELIMITER //

CREATE TRIGGER trg\_mentor\_data\_integrity

BEFORE UPDATE ON mentor

FOR EACH ROW

BEGIN

DECLARE is\_mentor\_change\_allowed INT;

-- Check if the mentor has active mentoring sessions

SELECT COUNT(\*) INTO is\_mentor\_change\_allowed

FROM mentor

WHERE faculty\_id = OLD.faculty\_id

AND fam\_date >= CURDATE();

IF is\_mentor\_change\_allowed > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot modify the mentor with active mentoring sessions.';

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_scholarship\_data\_integrity

BEFORE UPDATE ON scholarship

FOR EACH ROW

BEGIN

DECLARE is\_scholarship\_change\_allowed INT;

-- Check if the scholarship has been awarded to any student

SELECT COUNT(\*) INTO is\_scholarship\_change\_allowed

FROM scholarship

WHERE scholarship\_id = OLD.scholarship\_id

AND student\_id IS NOT NULL;

IF is\_scholarship\_change\_allowed > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot modify a scholarship that has been awarded to a student.';

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER after\_delete\_faculty

AFTER DELETE ON faculty

FOR EACH ROW

BEGIN

-- Update faculty\_id in mentor table for all records with the deleted faculty\_id

UPDATE mentor

SET faculty\_id = 'PESIM'

WHERE faculty\_id = OLD.faculty\_id;

END //

DELIMITER ;

CREATE TRIGGER after\_student\_delete\_enrols

AFTER DELETE ON student

FOR EACH ROW

DELETE FROM enrols WHERE SRN = OLD.SRN;

/\*trigger to assign student to a faculty for mentorship if a faculty is removed \*/

CREATE TRIGGER after\_faculty\_delete\_mentor

AFTER DELETE ON faculty

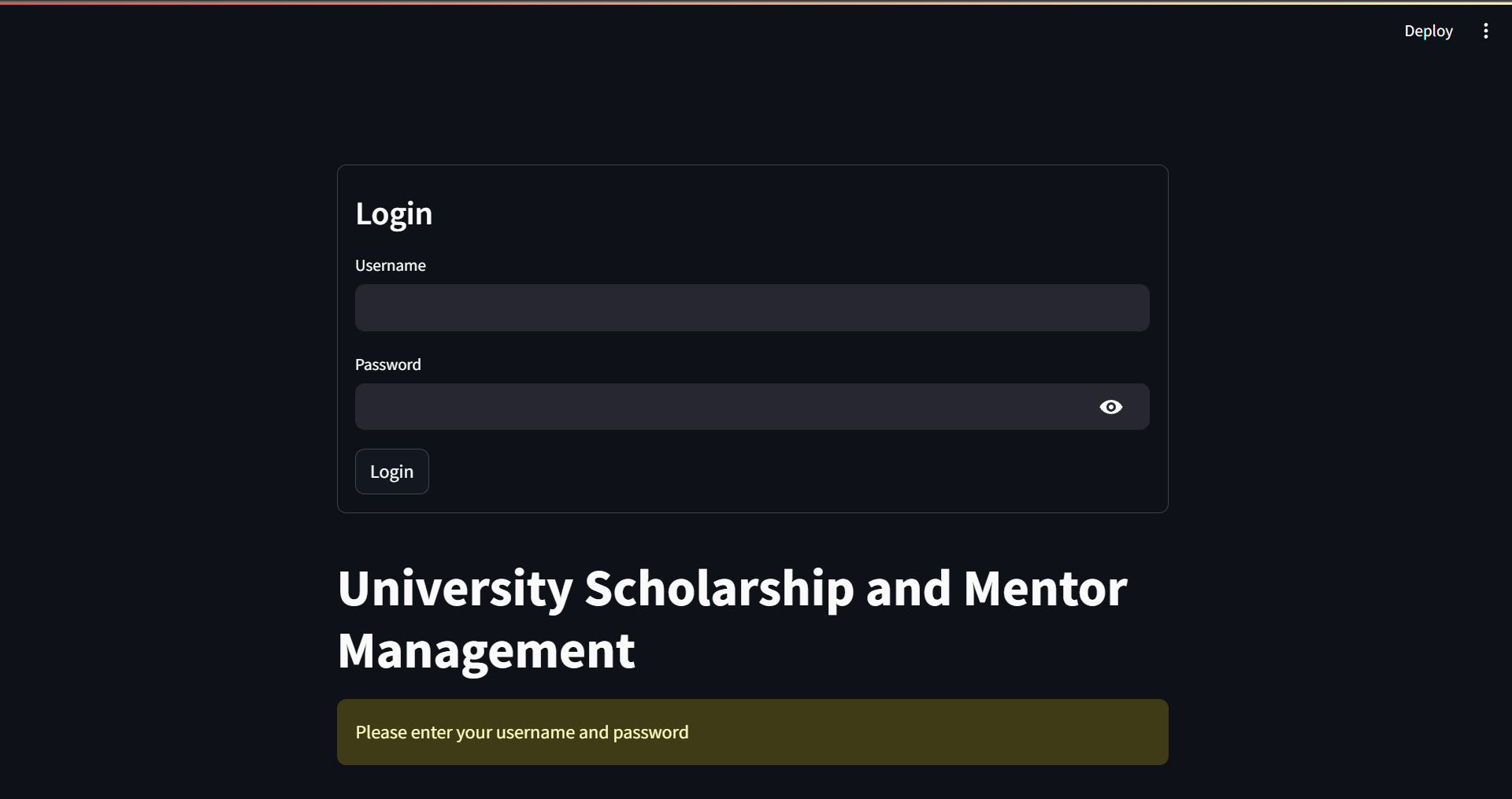
FOR EACH ROW

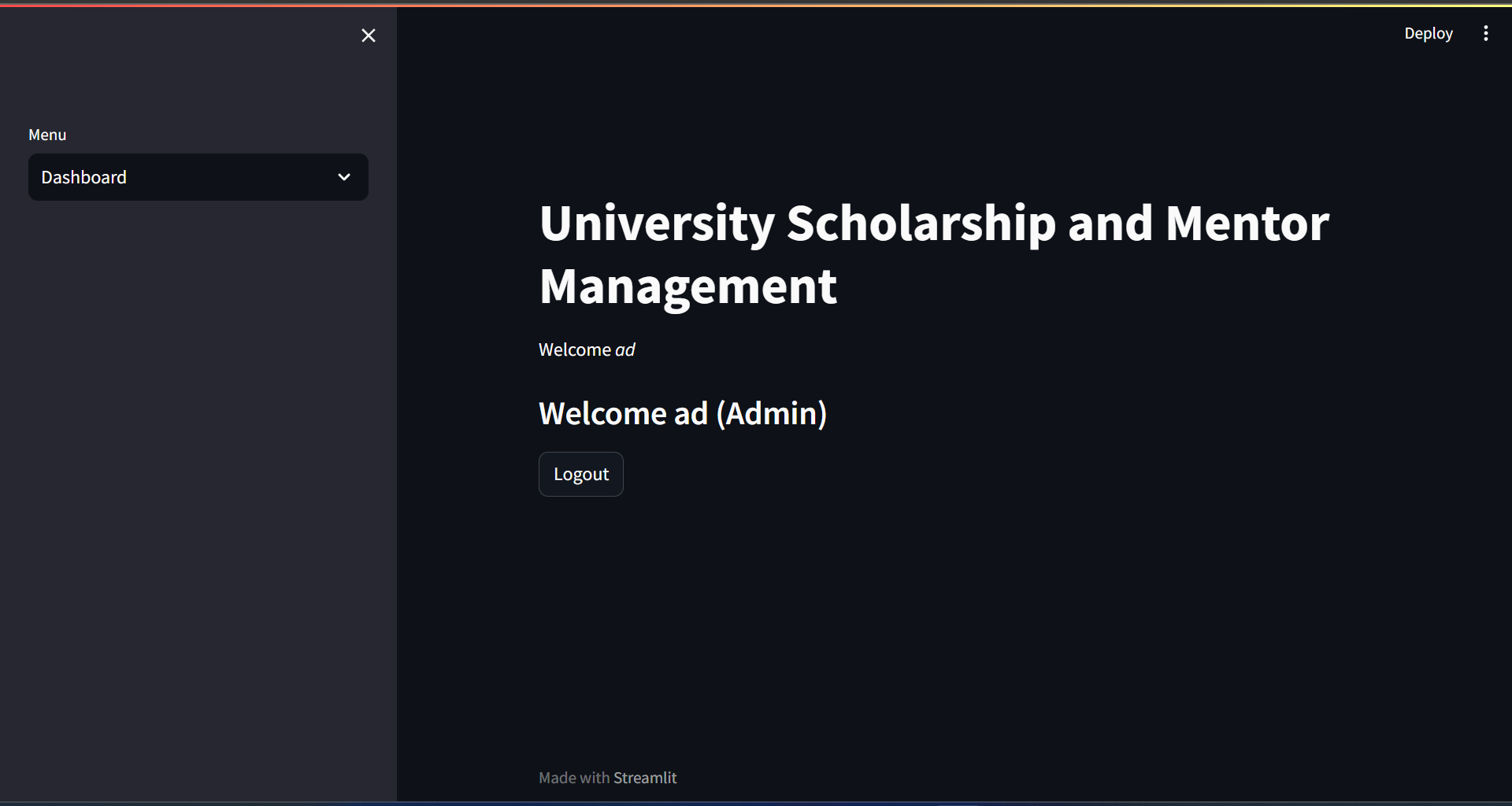
UPDATE mentor

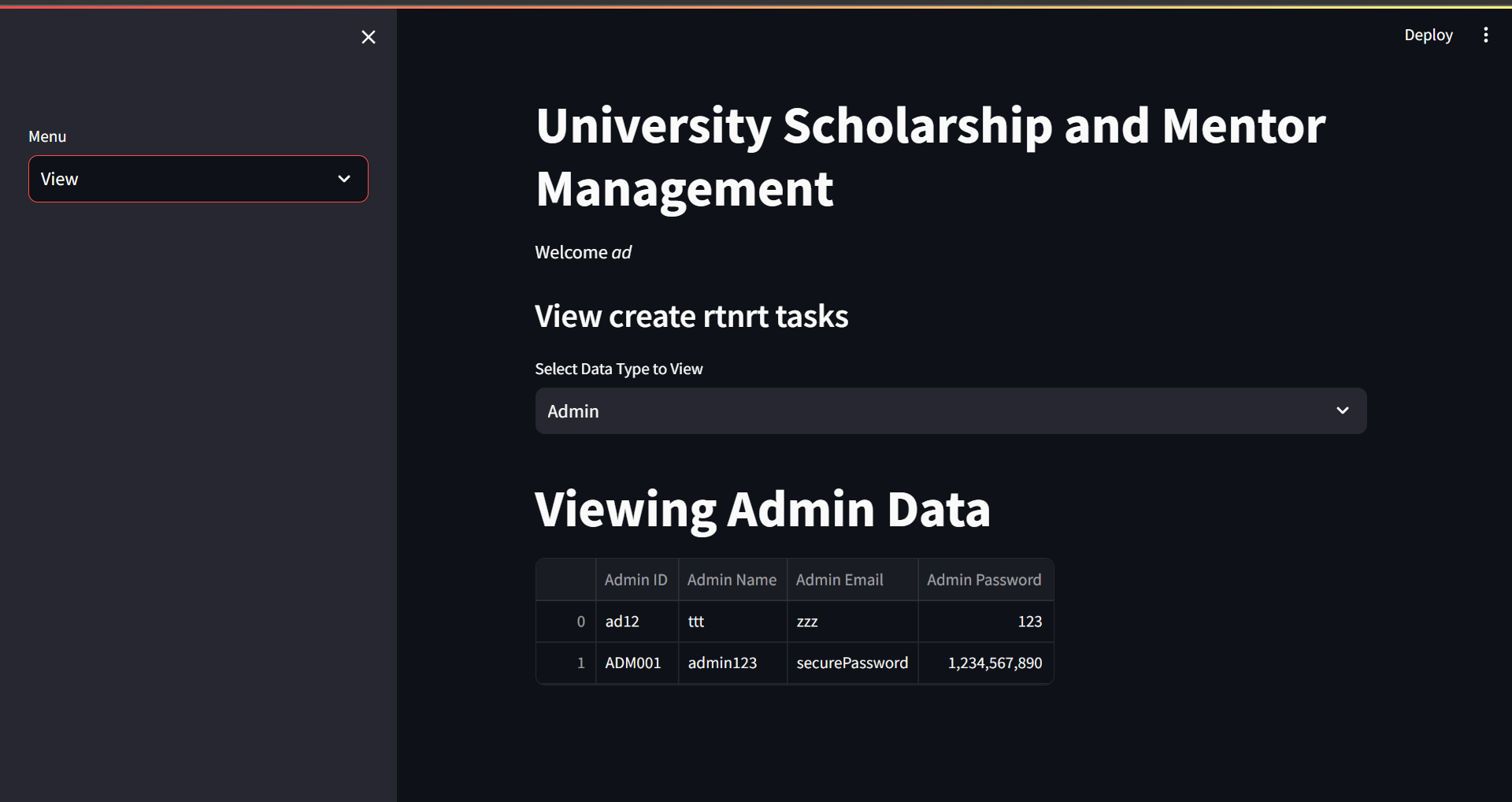
SET faculty\_id = 'PESIM'

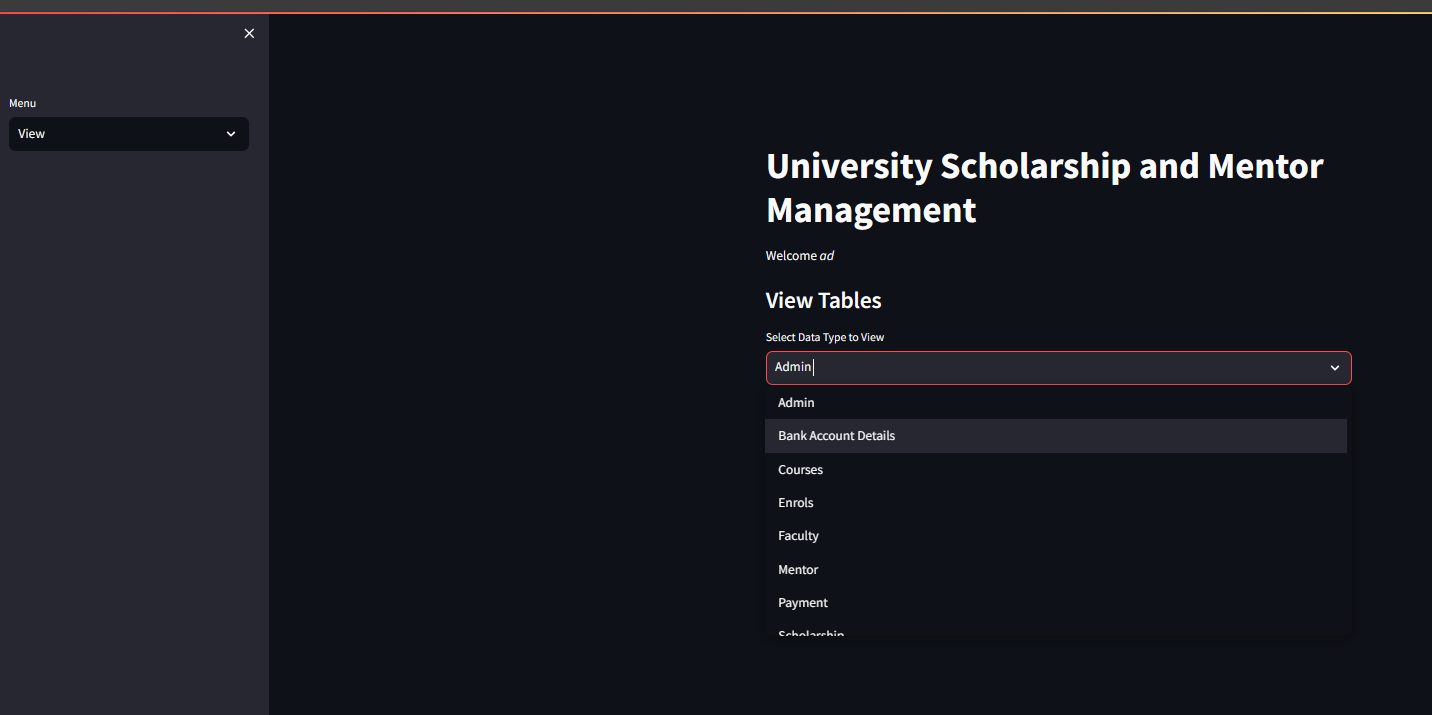
WHERE faculty\_id = OLD.faculty\_id;

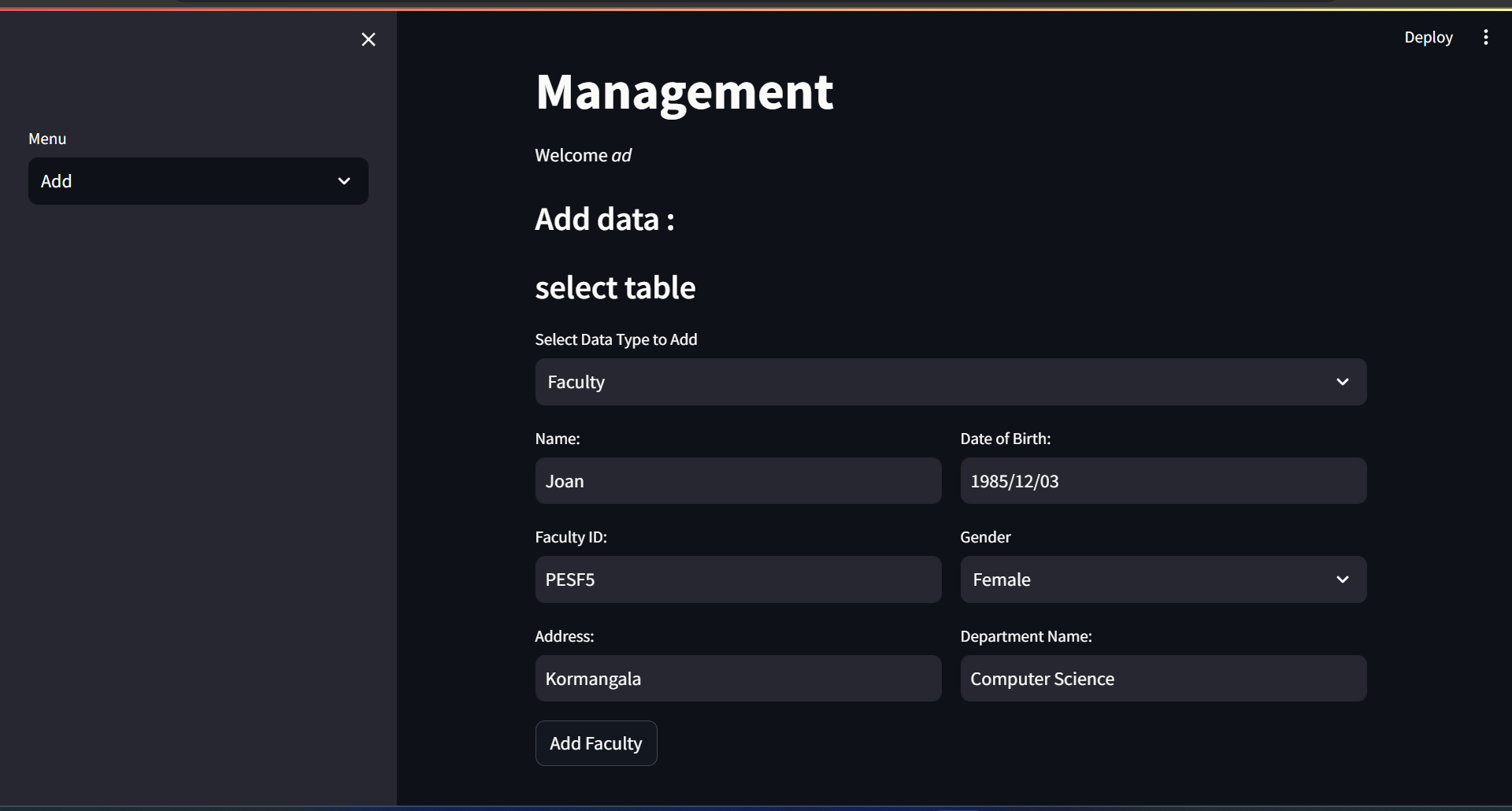
**9. FRONT END DEVELOPEMNT**

****

****

****

****

****

**REFERENCES**

[1] for Login Page[https://blog.streamlit.io/streamlit-authenticator-part-1-adding-an-authentication-component-to-your-app]

[2] for developing frontend [https://docs.streamlit.io]

[3] for database [https://dev.mysql.com/doc]

‘