DBMS PROJECT

ONLINE LEARNING PLATFORM

Pratheek Karike Venkatesha – PES1UG21CS444

Pruthvik L – PES1UG21CS456

Project Description

An online learning platform or E-Learning platform is a feature-rich web platform designed to help educational institutions with online learning, course administration, and assessment. It offers a centralised platform where educators, learners, and administrators can communicate with each other, oversee classes, carry out evaluations, and efficiently track advancement.

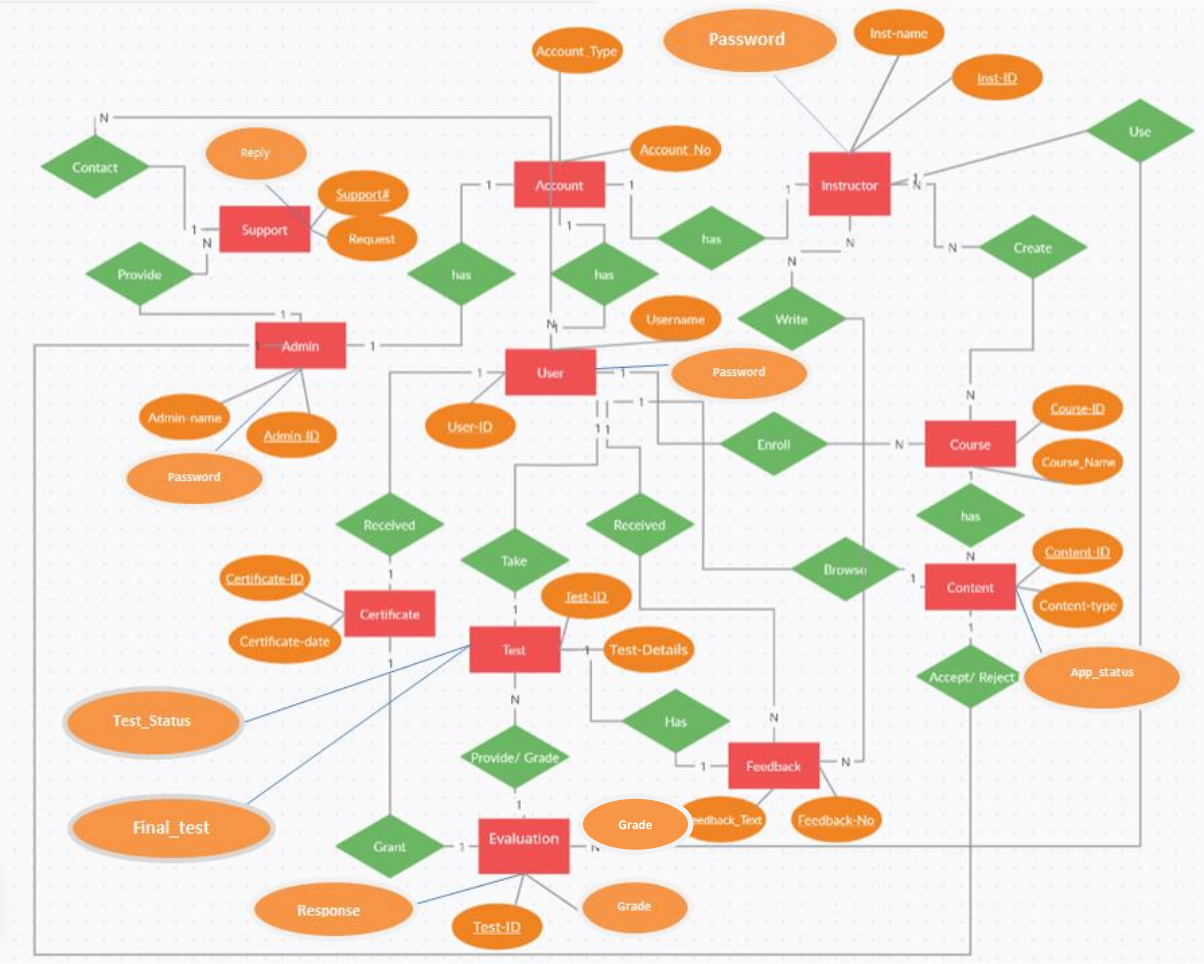
Key Features:

- User Roles: Distinct user roles for instructors, students, and administrators, each with specific permissions and functionalities.
- Course Creation and Management: Intuitive interfaces for instructors to create courses, upload content, and manage enrollments effortlessly.
- Assessment Tools: Robust assessment modules enabling instructors to create tests, evaluate submissions, and provide feedback efficiently along with certificates for course completion.
- Support System: A dedicated support system facilitating communication between users and administrators for query resolution.

Softwares/Tools/Programming languages

Software Tool - Microsoft Visual Studio Code, DBMS - MySQL 8.0, Languages & Frameworks - Python, Streamlit (Frontend)

ER Diagram


Account_Type
Password
Inst-name
Inst-ID
Use
Contact
Reply
Account_No
Account
Instructor
Support#
Support
Request
has
has
has
Create
Provide
Write
Admin
Username
User
Password
Admin-name
Admin-ID
User-ID
Enroll
Course
Course-ID
Course_Name
Password
Received
Received
has
Take
Browse
Content
Content-ID
Content-type
Certificate-ID
Certificate
Test-ID
Certificate-date
Test
Test-Details
Accept/ Reject
App_status
Test_Status
Has
Provide/ Grade
Feedback
Final_test
Grade
Feedback_Text
Feedback-No
Grant
Evaluation
Response
Test-ID
Grade

Relational Schema

User

User-ID	Username	Password

Account

Account#	Account-Type	User-ID	Instructor_ID

Instructor

Instructor-ID	Instructor-Name	Password

Course

Course-ID	Course-Name	Instructor-ID

Administrator

Admin-ID	Admin-Name	Password

Content

Content-ID	Content-Type	Course-ID	Admin-ID

Support

Admin-ID	Support#	Request	Reply	User-ID

Test

User-ID	Test-ID	Test-Details	Instructor-ID	Course-ID	Test-Status	Final-Test

Evaluation

Test-ID	Instructor-ID	Test-Grade	Test-Response	User-ID

Feedback

Test-ID	Feedback-No	Feedback-Text

Certificate

Certificate-ID	Test-ID	Certificate-Date	User-ID

DDL SQL Commands

Used for creating tables with constraints

```
# Create the Test table
cursor.execute('''
    CREATE TABLE IF NOT EXISTS Test (
        Test_ID INT AUTO_INCREMENT PRIMARY KEY,
        User_ID INT NOT NULL,
        Course_ID INT NOT NULL,
        Test_Details VARCHAR(255),
        Instructor_ID INT NOT NULL,
        Test_Status enum('Completed','Available') DEFAULT 'Available',
        final_test BOOLEAN NOT NULL,
        FOREIGN KEY (User_ID) REFERENCES User(User_ID),
        FOREIGN KEY (Course_ID) REFERENCES Course(Course_ID),
        FOREIGN KEY (Instructor_ID) REFERENCES Instructor(Instructor_ID)
    )
''')
```

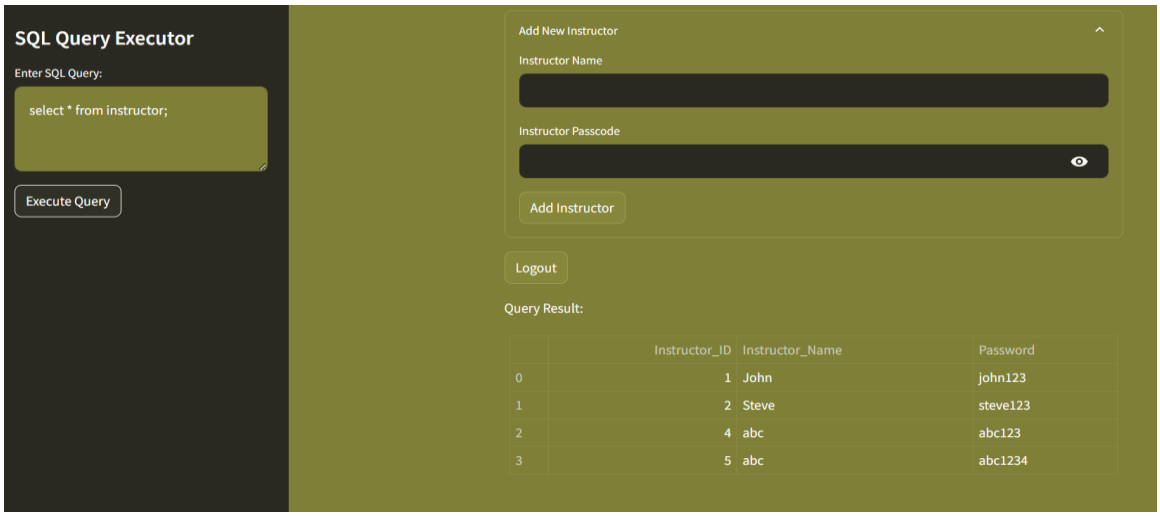
CRUD Operations

- **Create Operator: Used to create all 12 tables.**

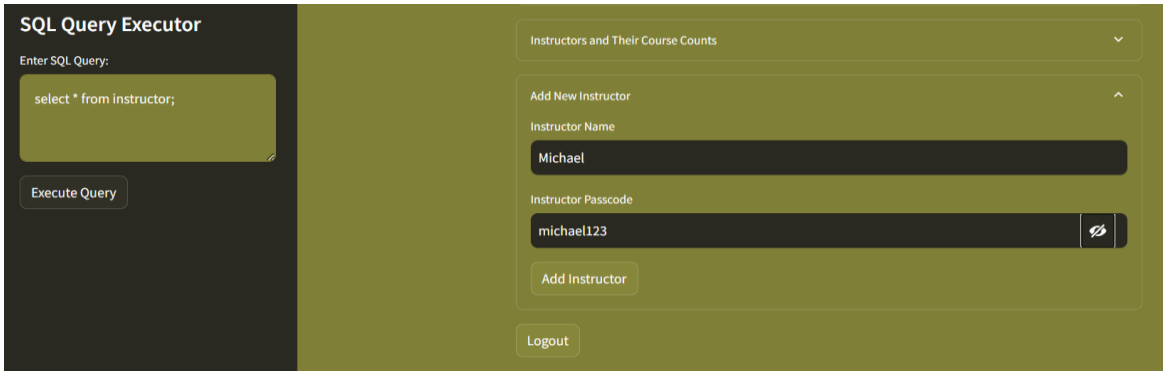
```
# Create the Account table
cursor.execute('''
    CREATE TABLE IF NOT EXISTS Account (
        Account_No INT AUTO_INCREMENT PRIMARY KEY,
        Account_Type ENUM('student','instructor','administrator') NOT NULL,
        User_ID INT,
        Instructor_ID INT,
        FOREIGN KEY(User_ID) REFERENCES User(User_ID),
        FOREIGN KEY(Instructor_ID) REFERENCES Instructor(Instructor_ID)
    )
''')
```

Values are inserted from frontend:

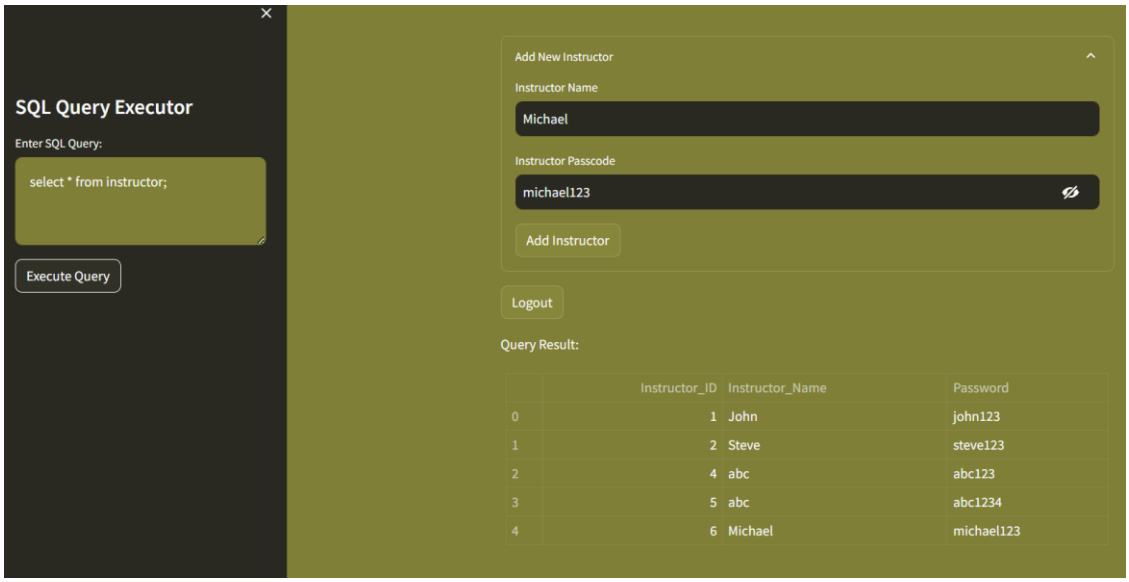
Instructor table before insertion:



Inserting values



Instructor table after insertion:



Similar insertion is done for all required tables (Course, User, Test etc.)

Example code snippet of Insert Query:

```python
def add_instructor(username, password):
    db = create_connection()
    cursor = db.cursor()

    # Insert into User table
    cursor.execute("INSERT INTO  Instructor (Instructor_Name, Password) VALUES (%s, %s)", (username, password))
    id = cursor.lastrowid

    # Insert into Account table with Account_Type 'instructor'
    cursor.execute("INSERT INTO Account (Account_Type,Instructor_ID) VALUES ('instructor',%s)", (id,))

    db.commit()
    cursor.close()
    db.close()
```

- **<u>Read Operator</u>: Select statements are used in various queries to display the required information.**

The below queries are used to fetch the number of students who have registered as well as number of instructors working.

```python
# Function to fetch the number of students
def fetch_student_count():
    db = create_connection()
    cursor = db.cursor(dictionary=True)  # Set dictionary=True to fetch results as dictionaries
    cursor.execute("SELECT COUNT(*) AS student_count FROM Account WHERE Account_Type = 'student'")
    student_count = cursor.fetchone()
    if student_count:
        student_count = student_count['student_count']
    else:
        student_count = 0  # Set default value if no count is returned
    db.close()
    return student_count

# Function to fetch the number of instructors
def fetch_instructor_count():
    db = create_connection()
    cursor = db.cursor(dictionary=True)  # Set dictionary=True to fetch results as dictionaries
    cursor.execute("SELECT COUNT(*) AS instructor_count FROM Account WHERE Account_Type = 'instructor'")
    instructor_count = cursor.fetchone()
    if instructor_count:
        instructor_count = instructor_count['instructor_count']
    else:
        instructor_count = 0  # Set default value if no count is returned
    db.close()
    return instructor_count
```

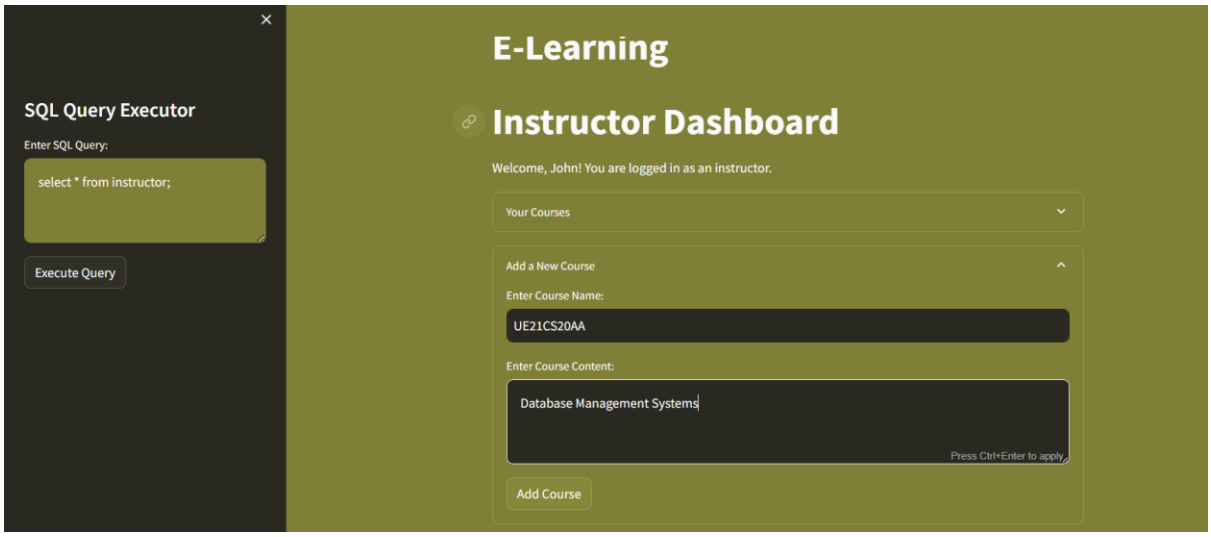
- <u>Update Operator</u>: **Many queries are used to update the contents of a table.**

The below query enables the admin to approve a course suggested by an instructor

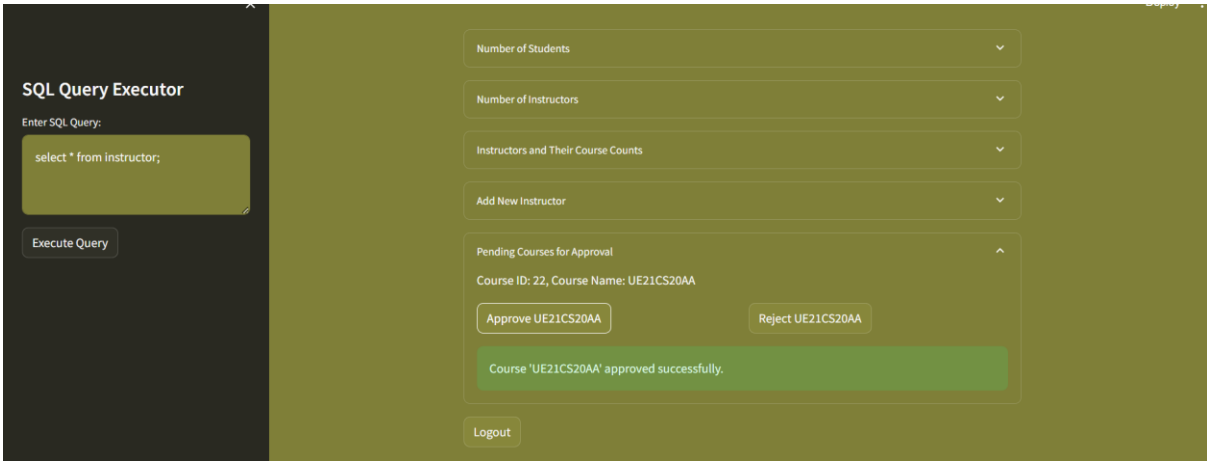
```
# Function to approve a pending course
def approve_course(course_id, admin_id):
    db = create_connection()
    cursor = db.cursor()
    cursor.execute("UPDATE Content SET Approval_Status = 'Approved' WHERE Course_ID = %s", (course_id,))
    cursor.execute("UPDATE Content SET Admin_ID = %s WHERE Course_ID = %s", (admin_id,course_id))

    db.commit()
    cursor.close()
    db.close()
```

An instructor can define a new course as shown below



Admins get an option to approve or reject a course whenever there is a pending course as shown below



The new course has been added successfully

Query Result:

	Course_ID	course_name	Instructor_ID
0	15	2023DAA210	2
1	16	2023SE215	2
2	17	2023LA123	1
3	18	2023MA123	1
4	20	2023DBMS210	2
5	22	UE21CS20AA	1

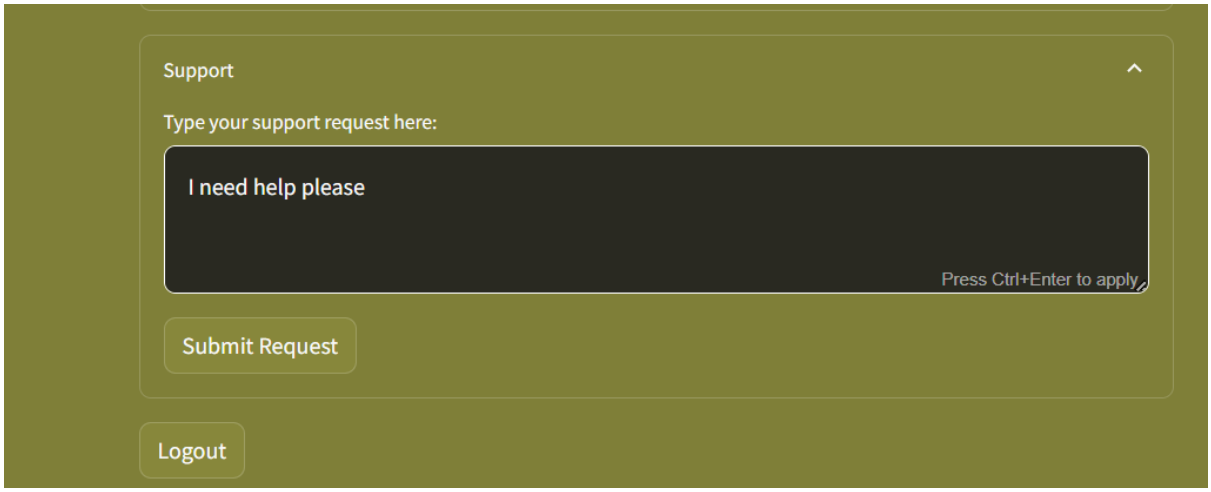
Similarly update operators have been used in other queries. One example is in grading a test that a student has submitted. The test_grade attribute is updated with the corresponding grade.

- **<u>Delete Operator</u>: Many queries are used to delete entries from a table.**

One example is a query to delete a support query after the issue has already been resolved by an admin.

```python
    # Function to delete a resolved support entry
def delete_support_entry(support_id):
    db = create_connection()
    cursor = db.cursor()
    cursor.execute("DELETE FROM Support WHERE Support_No = %s", (support_id,))
    db.commit()
    cursor.close()
    db.close()
```

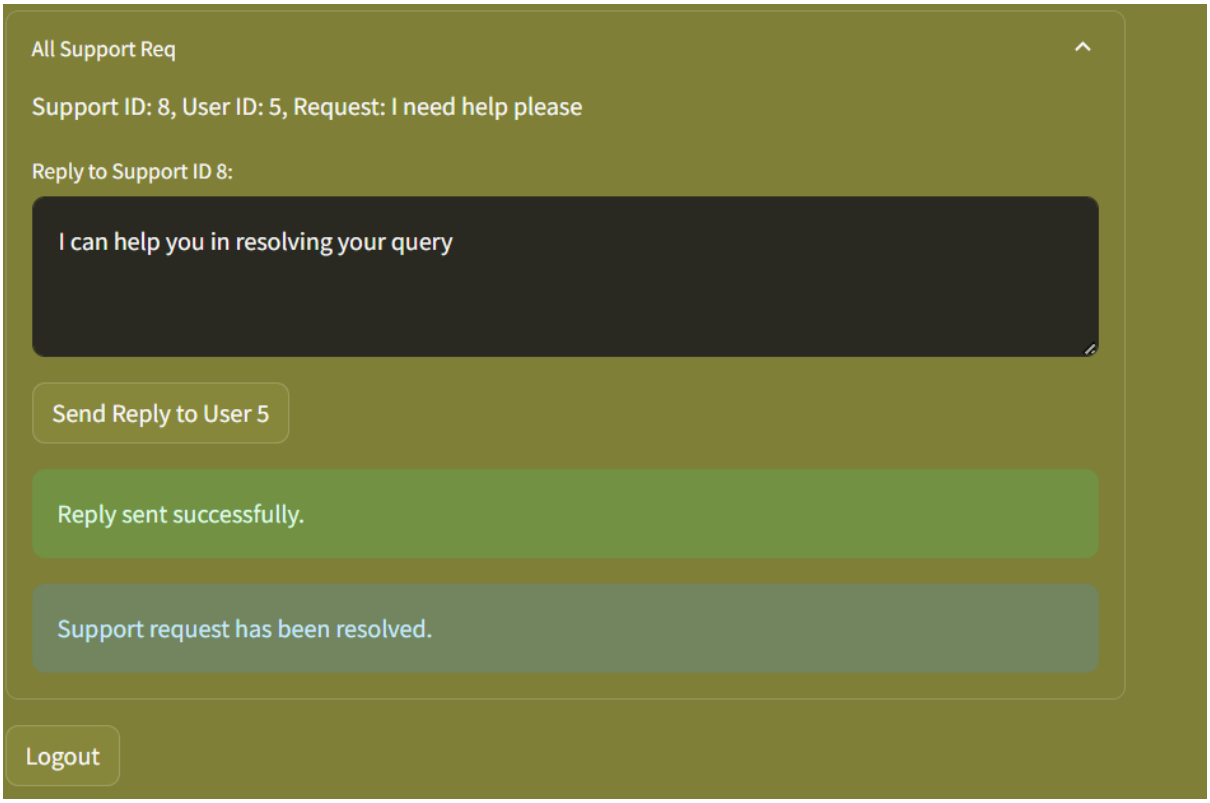
When a student asks for support, initially an entry is made into the support table with the corresponding request.



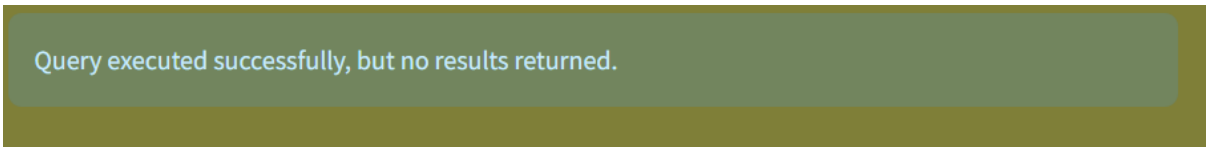
Query Result:

	Admin_ID	Support_No	Request	Reply	User_ID
0	2	8	I need help please	-	5

An admin can resolve the query, and the corresponding request will get deleted from the support table



The support table is empty when all pending support requests are resolved.



Similarly delete operator has been used in other queries such as in deleting a course.

Functionalities

User Authentication and Access Control

- **Login and Registration:**
  - **Users (students, instructors, administrators) can log in using their credentials.**
  - **New users can register to create accounts.**
  - **Role-based access control allows different functionalities based on user roles (admin, student, and instructor).**

Dashboard Functionalities

1. **Admin Dashboard:**
   - **Admins can:**
     - **View total counts of students and instructors.**
     - **List instructors with their course counts.**
     - **Add new instructors to the system.**
     - **Manage pending course approvals.**
     - **Respond to and resolve support requests.**

2. **Student Dashboard:**
   - **Students can:**
     - **View enrolled courses and their statuses (Completed or Ongoing).**
     - **Enroll in available courses.**
     - **Interact with available tests for enrolled courses.**
     - **Submit tests, view past grades, and feedback.**
     - **View earned certifications for completed courses.**
     - **Request support from administrators.**

3. **<u>Instructor Dashboard</u>:**

    - **Instructors can:**
        - **View courses and enrolled courses with status.**
        - **Enroll students in courses.**
        - **Request support.**
        - **Submit tests for evaluation.**
        - **Access past test grades and feedback.**
        - **Fetch completed courses along with certifications.**

<u>Database Operations</u>

- **Database Handling:**
    - **Contains functions to create and manage connections to the database.**
    - **Executes SQL queries related to user authentication, dashboard functionalities, course management, and support requests.**

<u>Application Structure and Integration</u>

- **Main Application Structure:**
    - **Includes the main app structure, routing, and integration of different dashboard functionalities based on user authentication.**
    - **Integrates different functionalities from admin, student, and instructor dashboards based on user roles and access permissions.**

<u>Overall Functionality</u>

- **User Management:**

    - **Enables users to access course-related functionalities based on their roles (admin, student, instructor).**

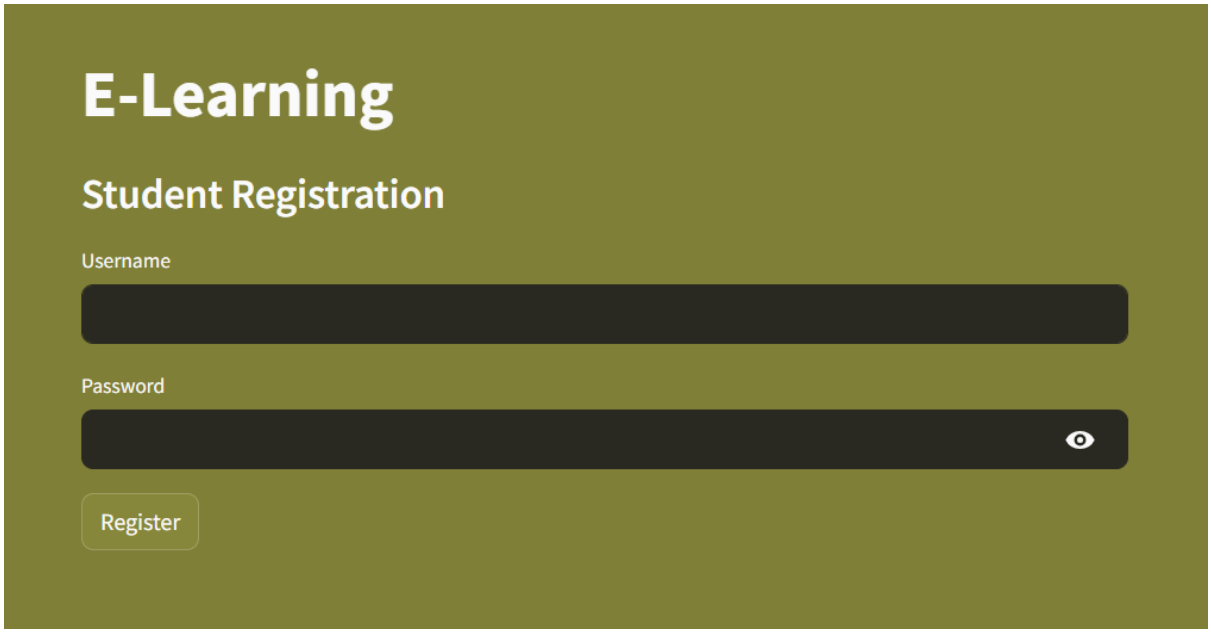
- **Course Enrollment and Management:**
  - **Allows users to enroll in courses, view course details, and manage enrolled courses.**

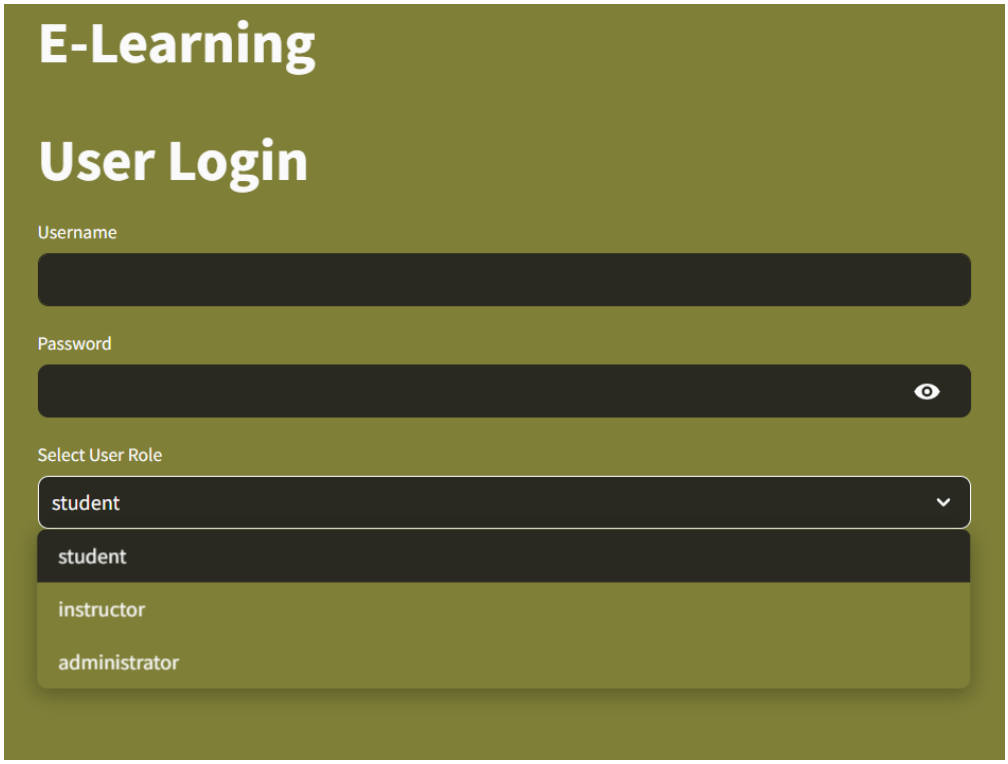
- **Test Submission and Evaluation:**
  - **Facilitates the submission of tests, evaluation, and retrieval of past grades and feedback.**

- **Support Request Handling:**
  - **Allows users to raise support requests and for admins to respond to and resolve them.**

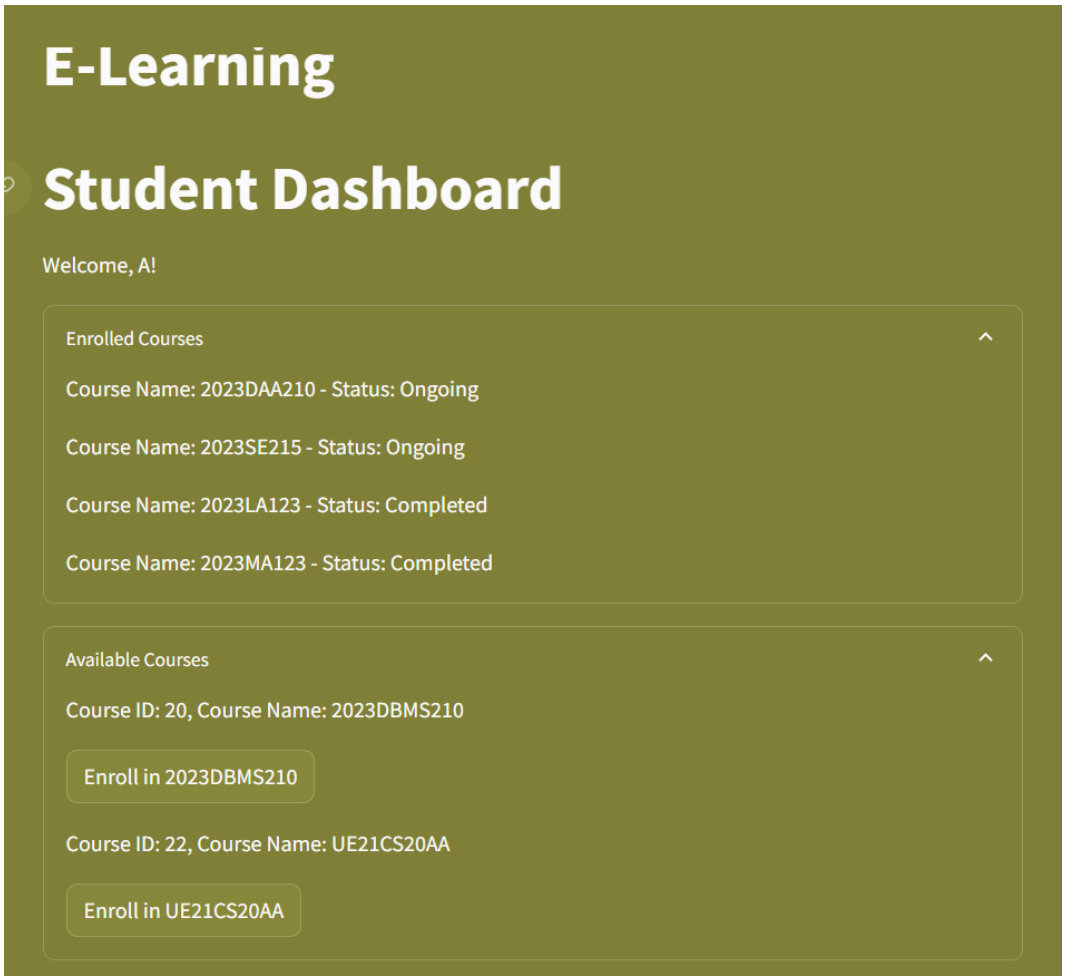
Frontend representation of functionalities

- **Registration & Login**





- **Student Dashboard**



Tests

Tests for 2023DAA210

Tests for 2023SE215

Tests for 2023LA123

Test ID: 19, Test Status: Available

Test Details: Test number 3

Your Test Response:

Submit Test

Tests for 2023MA123

Your Test Grades with Feedback:

Test ID: 12, Test Grade: A

Feedback: Nice

Test ID: 13, Test Grade: S

Feedback: vvwvwrgwrgwrf

Test ID: 14, Test Grade: B

Feedback: Good job

Test ID: 15, Test Grade: S

Feedback: Good

Test ID: 16, Test Grade: S

Feedback: Nice

Test ID: 17, Test Grade: A

Feedback: lfhqefle

Test ID: 18, Test Grade: S

Feedback: Good

Test ID: 19, Test Grade: None

Test Grades with Instructor Feedback

Your Test Grades with Feedback:

Test ID: 12, Test Grade: A

Feedback: Nice

Test ID: 13, Test Grade: S

Feedback: vvwvwrgwrgwrf

Test ID: 14, Test Grade: B

Feedback: Good job

Test ID: 15, Test Grade: S

Feedback: Good

Test ID: 16, Test Grade: S

Feedback: Nice

Test ID: 17, Test Grade: A

Feedback: lfhqefle

Test ID: 18, Test Grade: S

Feedback: Good

Certifications

Certificates Earned:

Certificate of Achievement for successfully clearing the course '2023LA123'

Certificate of Achievement for successfully clearing the course '2023MA123'

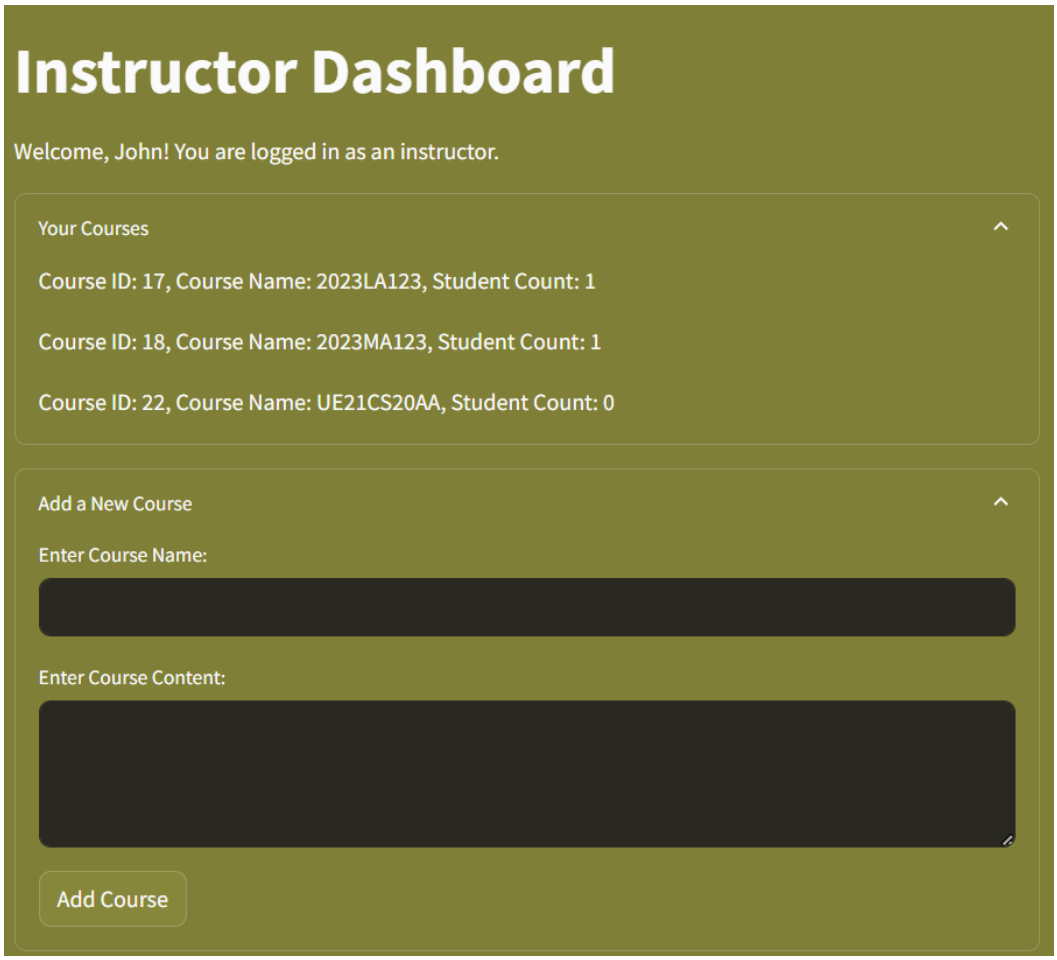
Support

Type your support request here:

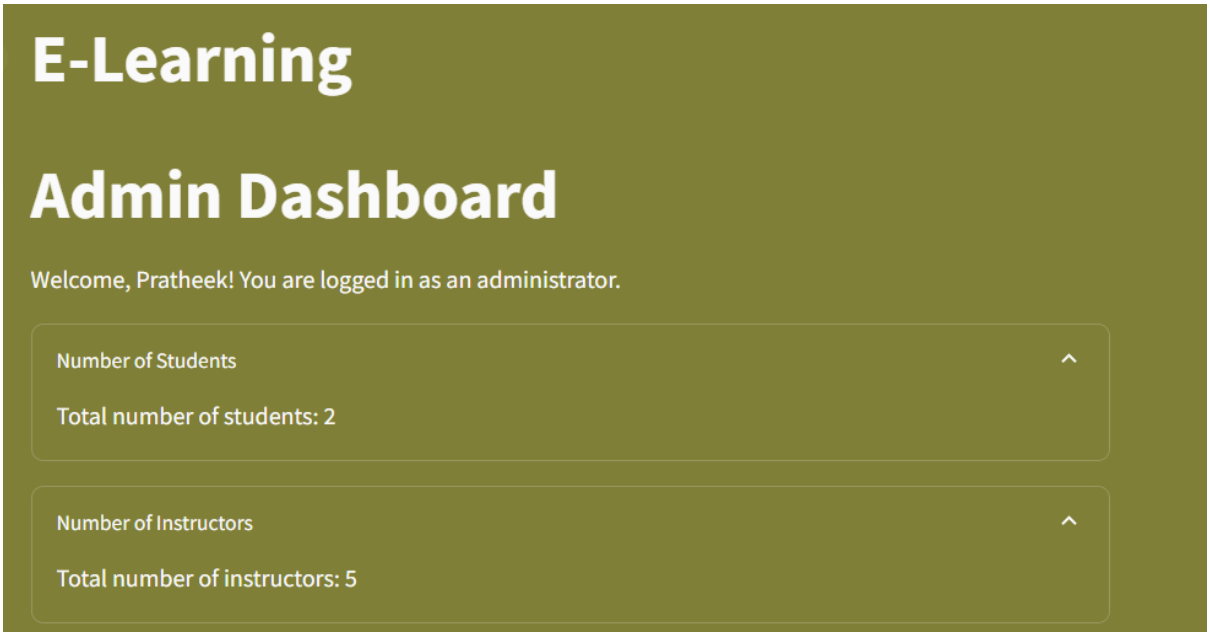
Submit Request

Logout

- **Instructor Dashboard**



- **Admin Dashboard**



Instructors and Their Course Counts

Instructor Name: John

Number of Courses: 3

Instructor Name: Steve

Number of Courses: 3

Instructor Name: abc

Number of Courses: 0

Instructor Name: Michael

Number of Courses: 0

Add New Instructor

Instructor Name

Instructor Passcode

Add Instructor

All Support Req

Support ID: 9, User ID: 5, Request: Hey I need help in resolving an issue.

Reply to Support ID 9:

Send Reply to User 5

Logout

Usage of Triggers

```
# Create the generate_certificate_after_grading trigger
cursor.execute('''
    DELIMITER //
    CREATE TRIGGER generate_certificate_after_grading
    AFTER UPDATE ON Evaluation
    FOR EACH ROW
    BEGIN
        DECLARE is_final_test INT;

        IF NEW.Test_Grade = 'S' THEN
            SELECT final_test INTO is_final_test FROM Test WHERE Test_ID = NEW.Test_ID;

            IF is_final_test = 1 THEN
                INSERT INTO Certificate (User_ID, Course_ID, Certificate_Date)
                VALUES (NEW.User_ID, (SELECT Course_ID FROM Test WHERE Test_ID = NEW.Test_ID), NOW());
            END IF;
        END IF;
    END //
    DELIMITER ;
''')
```

The above trigger is used to generate a certificate if the user completes a course. When creating a test, the instructor can decide whether it is the final test. If it is the final test and the student gets an 'S' grade, then a certification of course completion is generated for that user using the above trigger.

Course ID 17 has the name 2023LA123

Your Courses

Course ID: 17, Course Name: 2023LA123, Student Count: 1

Course ID: 18, Course Name: 2023MA123, Student Count: 1

If the instructor grades a new test of this course (Test ID 15 in the below image) with an 'S' grade, then a certificate gets generated.

Test ID: 15, Test Grade: S

Feedback: Good

Certifications

Certificates Earned:

Certificate of Achievement for successfully clearing the course '2023LA123'

Usage of Procedures & Functions

- **The below procedure is used to generate the number of students that are enrolled in a given course**

```
"""# Create the GetStudentCountForCourse procedure
cursor.execute('''
    DELIMITER //
    CREATE PROCEDURE GetStudentCountForCourse(IN courseID INT)
    BEGIN
        SELECT COUNT(*) AS student_count FROM Enrollment WHERE Course_ID = courseID;
    END //
    DELIMITER ;
''')
```

```python
def get_student_count_for_course(course_id):
    db = create_connection()
    cursor = db.cursor(dictionary=True)
    cursor.execute("CALL GetStudentCountForCourse(%s)", (course_id,))
    result = cursor.fetchone()
    cursor.close()
    db.close()
    return result['student_count'] if result else 0
```

Your Courses

Course ID: 17, Course Name: 2023LA123, Student Count: 1

Course ID: 18, Course Name: 2023MA123, Student Count: 1

Course ID: 22, Course Name: UE21CS20AA, Student Count: 0

- **The below function is used to get the number of courses that each instructor is taking.**

```python
# Create the GetCourseCountForInstructor Function
cursor.execute('''
    DELIMITER //

    CREATE FUNCTION GetCourseCountForInstructor(instructorID INT)
    RETURNS INT
    DETERMINISTIC
    READS SQL DATA
    BEGIN
        DECLARE course_count INT;
        SELECT COUNT(*) INTO course_count FROM Course WHERE Instructor_ID = instructorID;
        RETURN course_count;
    END //

    DELIMITER ;
''')"""
```

```python
def fetch_all_instructors_with_course_count():
    db = create_connection()
    cursor = db.cursor(dictionary=True)
    cursor.execute("""
        SELECT Instructor.*, GetCourseCountForInstructor(Instructor.Instructor_ID) AS Course_Count
        FROM Instructor
        INNER JOIN Account ON Instructor.Instructor_ID = Account.Instructor_ID
        WHERE Account.Account_Type = 'instructor'
    """)
    instructors = cursor.fetchall()
    db.close()
    return instructors
```

Instructors and Their Course Counts

Instructor Name: John

Number of Courses: 3

Instructor Name: Steve

Number of Courses: 3

Instructor Name: abc

Number of Courses: 0

Instructor Name: Michael

Number of Courses: 0

Usage of Nested queries, join and aggregate queries

- **Join queries**

The below query fetches all the courses that a student has enrolled in

```
def fetch_enrolled_courses_with_status(student_id):
    db = create_connection()
    cursor = db.cursor(dictionary=True)
    cursor.execute("""
        SELECT Course.Course_ID, Course.course_name, Enrollment.course_status
        FROM Course
        INNER JOIN Enrollment ON Course.Course_ID = Enrollment.Course_ID
        WHERE Enrollment.User_ID = %s
    """, (student_id,))
    enrolled_courses = cursor.fetchall()
    db.close()
    return enrolled_courses
```

Your Courses

Course ID: 17, Course Name: 2023LA123, Student Count: 1

Course ID: 18, Course Name: 2023MA123, Student Count: 1

Course ID: 22, Course Name: UE21CS20AA, Student Count: 0

The below query fetches test grades & feedback

```
# Function to fetch past test grades along with feedback for a specific student
def fetch_student_test_grades_with_feedback(user_id):
    db = create_connection()
    cursor = db.cursor(dictionary=True)
    cursor.execute("""
        SELECT Evaluation.Test_ID, Evaluation.Test_Grade, Feedback.Feedback_Text
        FROM Evaluation
        LEFT JOIN Feedback ON Evaluation.Test_ID = Feedback.Test_ID
        WHERE Evaluation.User_ID = %s
    """, (user_id,))
    past_test_grades_with_feedback = cursor.fetchall()
    db.close()
    return past_test_grades_with_feedback
```

Test Grades with Instructor Feedback

Your Test Grades with Feedback:

Test ID: 12, Test Grade: A

Feedback: Nice

Test ID: 13, Test Grade: S

Feedback: vvwvwrgwrgwrf

Test ID: 14, Test Grade: B

Feedback: Good job

Test ID: 15, Test Grade: S

Feedback: Good

Test ID: 16, Test Grade: S

Feedback: Nice

The below query fetches completed tests

```
# Function to fetch completed tests for a specific user
def fetch_completed_tests(user_id):
    db = create_connection()
    cursor = db.cursor(dictionary=True)
    cursor.execute("SELECT * FROM Test INNER JOIN Evaluation ON Test.Test_ID = Evaluation.Test_ID "
                   "WHERE Evaluation.User_ID = %s AND Test.Test_Status = 'Completed'", (user_id,))
    completed_tests = cursor.fetchall()
    db.close()
    return completed_tests
```

The below query fetches the pending tests for an instructor to evaluate. This is also a nested query.

```python
# Modify the function to fetch pending tests for evaluation based on submitted responses
def fetch_pending_tests_for_evaluation(instructor_id):
    db = create_connection()
    cursor = db.cursor(dictionary=True)
    cursor.execute("""
        SELECT Evaluation.Test_ID, Evaluation.User_ID
        FROM Evaluation
        INNER JOIN Test ON Evaluation.Test_ID = Test.Test_ID
        WHERE Evaluation.Instructor_ID = %s
        AND Evaluation.Test_Grade IS NULL
        AND Test.Test_Status = 'Completed'
        AND Evaluation.Test_ID IN (
            SELECT Test_ID
            FROM Evaluation
            WHERE Test_Response IS NOT NULL
        )
    """, (instructor_id,))
    pending_tests = cursor.fetchall()
    db.close()
    return pending_tests
```

The below queries fetch pending courses for approval & instructor course count

```
def fetch_all_instructors_with_course_count():
    db = create_connection()
    cursor = db.cursor(dictionary=True)
    cursor.execute("""
        SELECT Instructor.*, GetCourseCountForInstructor(Instructor.Instructor_ID) AS Course_Count
        FROM Instructor
        INNER JOIN Account ON Instructor.Instructor_ID = Account.Instructor_ID
        WHERE Account.Account_Type = 'instructor'
    """)
    instructors = cursor.fetchall()
    db.close()
    return instructors

# Function to fetch pending courses for approval
def fetch_pending_courses():
    db = create_connection()
    cursor = db.cursor(dictionary=True)
    cursor.execute("""
    SELECT Course.*
    FROM Course
    JOIN Content ON Course.Course_ID = Content.Course_ID
    WHERE Content.Approval_Status = 'Pending'
""")
```

All of the above queries are some examples of join being used

- **<u>Nested Queries</u>**

Firstly, the query used previously to fetch pending evaluations is a nested query

The below query is used to get all available courses that a student hasn't enrolled in

```
# Function to fetch all courses excluding enrolled courses for a student
def fetch_all_courses_exclude_enrolled(student_id):
    db = create_connection()
    cursor = db.cursor(dictionary=True)
    cursor.execute("""
        SELECT * FROM Course WHERE Course_ID NOT IN
        (SELECT Course_ID FROM Enrollment WHERE User_ID = %s)
    """, (student_id,))
    courses = cursor.fetchall()
    db.close()
    return courses
```

Available Courses

Course ID: 20, Course Name: 2023DBMS210

Enroll in 2023DBMS210

Course ID: 22, Course Name: UE21CS20AA

Enroll in UE21CS20AA

The next query is used to display the feedback a student got for their test

```
 feedback for a specific test and user
test_id, user_id):
nection()
sor(dictionary=True)
"SELECT Feedback_Text FROM Feedback WHERE Test_ID = %s AND Test_ID IN (SELECT Test_ID FROM Evaluation WHERE User_ID = %s)", (test_id, user
or.fetchone()
```

Test Grades with Instructor Feedback

Your Test Grades with Feedback:

Test ID: 12, Test Grade: A

Feedback: Nice

The next query is used to display all the tests that were created by an instructor

```
ests for a specific instructor
tests(instructor_id):
ction()
r(dictionary=True)
ELECT * FROM Test WHERE User_ID IN (SELECT Instructor_ID FROM Account WHERE Account_Type = 'instructor' AND Instructor_ID = %s)", (instruc
tchall()
```

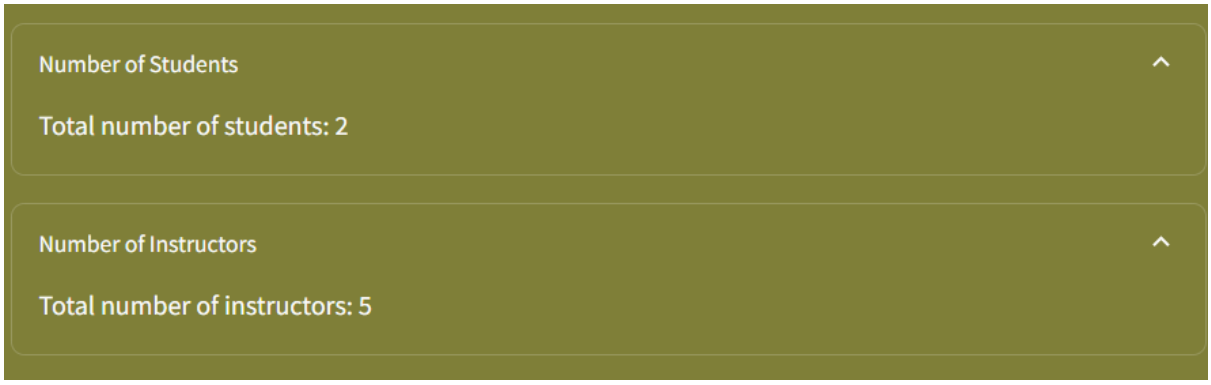
This is useful when fetching the pending evaluations

- **<u>Aggregate Queries</u>**

The below 2 queries are used to fetch the total student registration count and instructor count

```python
# Function to fetch the number of students
def fetch_student_count():
    db = create_connection()
    cursor = db.cursor(dictionary=True)  # Set dictionary=True to fetch results as dictionaries
    cursor.execute("SELECT COUNT(*) AS student_count FROM Account WHERE Account_Type = 'student'")
    student_count = cursor.fetchone()
    if student_count:
        student_count = student_count['student_count']
    else:
        student_count = 0  # Set default value if no count is returned
    db.close()
    return student_count

# Function to fetch the number of instructors
def fetch_instructor_count():
    db = create_connection()
    cursor = db.cursor(dictionary=True)  # Set dictionary=True to fetch results as dictionaries
    cursor.execute("SELECT COUNT(*) AS instructor_count FROM Account WHERE Account_Type = 'instructor'")
    instructor_count = cursor.fetchone()
    if instructor_count:
        instructor_count = instructor_count['instructor_count']
    else:
        instructor_count = 0  # Set default value if no count is returned
    db.close()
    return instructor_count
```



Summary

An online learning or E-Learning platform is a full-featured educational system made to meet the various demands of educators, administrators, and students. Its features include user authentication, role-based dashboards, course management, assignment submissions, and support request handling. It was developed with an emphasis on usability and functionality.

Throughout the project, satisfying user needs with powerful functionalities and intuitive interfaces was emphasized. The secret to the platform's success is its capacity to expedite the learning process, promote efficient communication, and offer a smooth experience to all parties involved.

There is a lot of scope to build on this project and add more functionalities, however we believe that this is a good basis with many fundamental concepts of DBMS applied successfully.