Mini project report on

Smart E-learning Platform Database Management System

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering

UE22CS351A – DBMS Project

Submitted by:

Sreeya Chatterjee	**PES2UG22CS574**
Sudhanshu Singh	**PES2UG22CS586**

Under the guidance of
Prof. Nivedita Kasturi
Assistant Professor
PES University

AUG - DEC 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

CERTIFICATE

This is to certify that the mini project entitled

Smart E-Learning Platform

is a bonafide work carried out by

Sreeya Chatterjee	**PES2UG22CS574**
Sudhanshu Singh	**PES2UG22CS586**

In partial fulfilment for the completion of fifth semester DBMS Project (UE22CS351A) in the Program of Study - Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period AUG. 2024 – DEC. 2024. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The project has been approved as it satisfies the 5th semester academic requirements in respect of project work.

Signature
Prof. Nivedita Kasturi
Assistant Professor

DECLARATION

We hereby declare that the DBMS Project entitled **Smart E-Learning Platform** has been carried out by us under the guidance of **Prof. Nivedita Kasturi, Assistant Professor** and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester AUG – DEC 2024.

Sreeya Chatterjee	**PES2UG22CS574**
Sudhanshu Singh	**PES2UG22CS586**

ABSTRACT

This project presents the development of a Smart E-learning Platform Database Management System (DBMS) designed to manage essential academic and administrative data for a digital education environment. The system includes nine core entities: `student`, `instructor`, `course`, `enrolment`, `assignment`, `quiz`, `submission`, `complaint`, and `phone`. Each entity is modelled as a table in the database to store structured data, enabling efficient storage, retrieval, and manipulation of information related to users, course content, assessments, and communications within the platform.

The platform provides comprehensive CRUD (Create, Read, Update, Delete) functionalities through an intuitive frontend interface built with 'Streamlit', allowing for seamless interaction with the database. Advanced features include tracking course enrolments, managing assignment submissions, recording student progress, and handling complaints, ensuring that both academic and support needs are addressed. This setup leverages MySQL for robust data management, allowing for complex relational operations and queries that are integral to educational platforms.

Overall, the Smart E-learning Platform DBMS offers a scalable, user-friendly solution for educational institutions to streamline data operations, enhance student-instructor interactions, and promote a structured digital learning experience.

TABLE OF CONTENTS

INTRODUCTION

The Smart E-Learning Platform is a comprehensive digital tool designed to enhance online learning experiences for students, instructors, and administrators. This platform allows for seamless management of various academic activities, including assignment submissions, quiz participation, course enrolments, and complaint handling, ensuring efficient communication across all user roles. Through an organized and scalable database management system, it securely stores and manages extensive educational data, enabling administrators and educators to oversee and adapt to students' learning needs effectively.

The platform’s structured approach simplifies the organization of large datasets, supporting educational institutions in streamlining workflows and maintaining records with accuracy. With a user-friendly interface, the platform promotes an interactive learning environment that encourages student engagement while allowing instructors to manage courses, assignments, and assessments with ease. Developed with modern database technologies and intuitive front-end tools, the Smart E-Learning Platform is tailored to support the evolving demands of today’s educational landscape, offering a dynamic solution adaptable to a variety of academic settings.

PROBLEM DEFINITION WITH USER REQUIREMENT SPECIFICATIONS

Traditional learning management systems often struggle to keep up with the growing volume of student data, resulting in difficulties with record maintenance and limitations in providing instant access to learning resources. The Smart E-Learning Platform addresses these challenges by introducing a centralized, efficient database that enables fast, secure storage and retrieval of critical information. By streamlining data handling, this platform reduces the administrative load on educators and administrators while enhancing the overall learning experience for students.

The primary user requirements for this platform include the following:

- **Students**: Require the ability to register for an account, enrol in courses, submit assignments, view quiz results, and file complaints or requests. The platform should offer students a simple, intuitive interface to access resources and track their academic progress in real time.

- **Instructors**: Need effective tools to manage their courses, create quizzes, grade assignments, and respond to student inquiries or complaints. These tools should be flexible, allowing instructors to update course materials and assignments while monitoring student performance and engagement.

- **Administrators**: Require comprehensive control over platform activities, including user role management, data oversight, and report generation. Administrators need streamlined features to monitor platform usage, handle user accounts, and compile data for institutional reporting purposes.

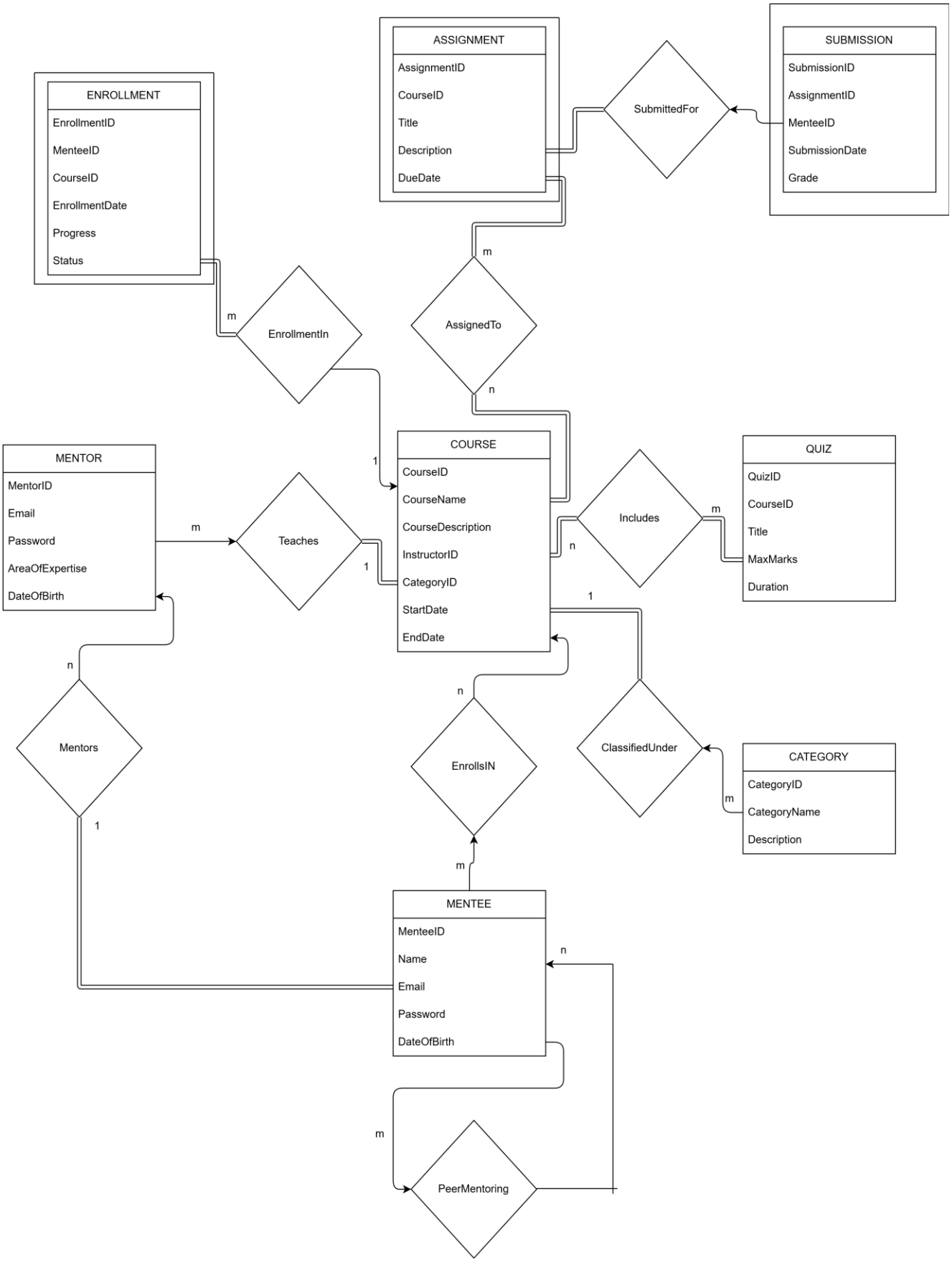
This project aims to implement these user requirements through a structured, scalable, and reliable approach, ensuring that each user role is empowered to perform their tasks efficiently within a single, cohesive platform. The Smart E-Learning Platform is built to adapt to evolving educational needs, making it a valuable asset for modern educational institutions.

LIST OF SOFTWARES/TOOLS/PROGRAMMING LANGUAGES USED

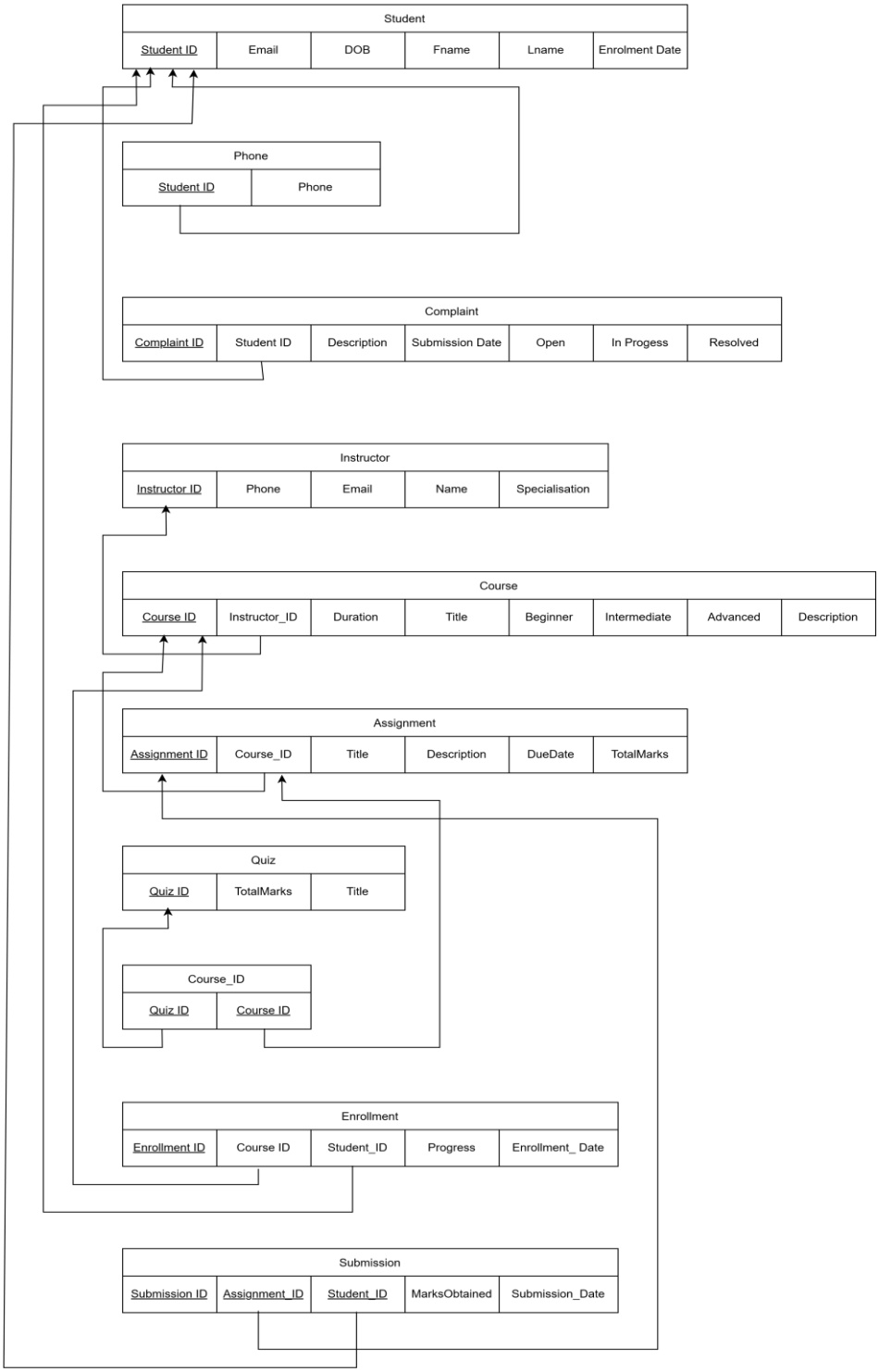
For the Smart E-Learning Platform, we employed a range of tools:

- **Database**: MySQL for managing the backend database due to its reliability, support for SQL standards, and scalability.
- **Programming Language**: Python, chosen for its simplicity and rich ecosystem of libraries.
- **Framework**: Streamlit, providing a web-based interface for CRUD operations and data visualization.
- **Additional Tools**: SQLAlchemy for database interactions, and IDEs like Visual Studio Code for development. Each tool contributes to an efficient development process, enabling robust data management and a seamless user experience.

ER MODEL


ASSIGNMENT
AssignmentID
CourseID
Title
Description
DueDate
SUBMISSION
SubmissionID
AssignmentID
MenteeID
SubmissionDate
Grade
SubmittedFor
ENROLLMENT
EnrollmentID
MenteeID
CourseID
EnrollmentDate
Progress
Status
EnrollmentIn
AssignedTo
COURSE
CourseID
CourseName
CourseDescription
InstructorID
CategoryID
StartDate
EndDate
MENTOR
MentorID
Email
Password
AreaOfExpertise
DateOfBirth
Teaches
Includes
QUIZ
QuizID
CourseID
Title
MaxMarks
Duration
Mentors
EnrollsIN
ClassifiedUnder
CATEGORY
CategoryID
CategoryName
Description
MENTEE
MenteeID
Name
Email
Password
DateOfBirth
PeerMentoring
m
n
1

ER TO RELATIONAL MAPPING



DDL STATEMENTS

```sql
CREATE TABLE assignment (
    AssignmentID INT NOT NULL PRIMARY KEY,
    CourseID INT,
    Title VARCHAR(100),
    Description TEXT,
    DueDate DATE,
    TotalMarks INT,
    FOREIGN KEY (CourseID) REFERENCES course(CourseID)
);

CREATE TABLE complaint (
    ComplaintID INT NOT NULL PRIMARY KEY,
    StudentID INT,
    Description TEXT,
    SubmissionDate DATE,
    Open TINYINT(1),
    InProgress TINYINT(1),
    Resolved TINYINT(1),
    FOREIGN KEY (StudentID) REFERENCES student(StudentID)
);
```

```sql
CREATE TABLE course (
    CourseID INT NOT NULL PRIMARY KEY,
    InstructorID INT,
    Duration INT,
    Title VARCHAR(100),
    Beginner TINYINT(1),
    Intermediate TINYINT(1),
    Advanced TINYINT(1),
    Description TEXT,
    FOREIGN KEY (InstructorID) REFERENCES instructor(InstructorID)
);

CREATE TABLE enrollment (
    EnrollmentID INT NOT NULL PRIMARY KEY,
    CourseID INT,
    StudentID INT,
    Progress VARCHAR(50),
    EnrollmentDate DATE,
    FOREIGN KEY (CourseID) REFERENCES course(CourseID),
    FOREIGN KEY (StudentID) REFERENCES student(StudentID)
);
```

```sql
CREATE TABLE instructor (
    InstructorID INT NOT NULL PRIMARY KEY,
    Phone VARCHAR(15),
    Email VARCHAR(255),
    Name VARCHAR(100),
    Specialisation VARCHAR(100)
);

CREATE TABLE phone (
    StudentID INT NOT NULL,
    Phone VARCHAR(15) NOT NULL,
    PRIMARY KEY (StudentID, Phone),
    FOREIGN KEY (StudentID) REFERENCES student(StudentID)
);
CREATE TABLE quiz (
    QuizID INT NOT NULL PRIMARY KEY,
    CourseID INT,
    Title VARCHAR(100),
    TotalMarks INT,
    FOREIGN KEY (CourseID) REFERENCES course(CourseID)
);
```

```sql
CREATE TABLE student (
    StudentID INT NOT NULL PRIMARY KEY,
    Email VARCHAR(255),
    DOB DATE,
    Fname VARCHAR(50),
    Lname VARCHAR(50),
    EnrollmentDate DATE
);

CREATE TABLE submission (
    SubmissionID INT NOT NULL PRIMARY KEY,
    AssignmentID INT,
    StudentID INT,
    MarksObtained INT,
    SubmissionDate DATE,
    FOREIGN KEY (AssignmentID) REFERENCES assignment(AssignmentID),
    FOREIGN KEY (StudentID) REFERENCES student(StudentID)
);
```

DML STATEMENTS (CRUD OPERATION SCREENSHOTS)

```python
# CRUD Functions for Student
def add_student_via_procedure(student_id, email, dob, fname, lname, enrollment_date):
    conn = create_connection()
    cursor = conn.cursor()
    query = "CALL add_student_procedure(%s, %s, %s, %s, %s, %s);"
    cursor.execute(query, (student_id, email, dob, fname, lname, enrollment_date))
    conn.commit()
    conn.close()

def view_students():
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM student")
    rows = cursor.fetchall()
    cursor.close()
    conn.close()
    return rows

def update_student(student_id, email, dob, fname, lname, enrollment_date):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("UPDATE student SET Email=%s, DOB=%s, Fname=%s, Lname=%s, EnrollmentDate=%s WHERE StudentID=%s",
                   (email, dob, fname, lname, enrollment_date, student_id))
    conn.commit()
    cursor.close()
    conn.close()

def delete_student(student_id):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM student WHERE StudentID = %s", (student_id,))
    conn.commit()
    cursor.close()
    conn.close()
```

```python
# CRUD Functions for Instructor
def add_instructor(instructor_id, phone, email, name, specialization):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("INSERT INTO instructor (InstructorID, Phone, Email, Name, Specialisation) VALUES (%s, %s, %s, %s, %s)",
                   (instructor_id, phone, email, name, specialization))
    conn.commit()
    cursor.close()
    conn.close()

def view_instructors():
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM instructor")
    rows = cursor.fetchall()
    cursor.close()
    conn.close()
    return rows

def update_instructor(instructor_id, phone, email, name, specialization):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("UPDATE instructor SET Phone=%s, Email=%s, Name=%s, Specialisation=%s WHERE InstructorID=%s",
                   (phone, email, name, specialization, instructor_id))
    conn.commit()
    cursor.close()
    conn.close()

def delete_instructor(instructor_id):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM instructor WHERE InstructorID = %s", (instructor_id,))
    conn.commit()
    cursor.close()
    conn.close()
```

```python
# CRUD Functions for Course
def add_course(course_id, instructor_id, duration, title, beginner, intermediate, advanced, description):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("INSERT INTO course (CourseID, InstructorID, Duration, Title, Beginner, Intermediate, Advanced, Description) VALUES (%s,
                   (course_id, instructor_id, duration, title, beginner, intermediate, advanced, description))
    conn.commit()
    cursor.close()
    conn.close()

def view_courses():
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM course")
    rows = cursor.fetchall()
    cursor.close()
    conn.close()
    return rows

def update_course(course_id, instructor_id, duration, title, beginner, intermediate, advanced, description):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("UPDATE course SET InstructorID=%s, Duration=%s, Title=%s, Beginner=%s, Intermediate=%s, Advanced=%s, Description=%s WHE
                   (instructor_id, duration, title, beginner, intermediate, advanced, description, course_id))
    conn.commit()
    cursor.close()
    conn.close()
```

```python
def delete_course(course_id):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM course WHERE CourseID = %s", (course_id,))
    conn.commit()
    cursor.close()
    conn.close()
```

```python
# CRUD Functions for Assignment
def add_assignment(assignment_id, course_id, title, description, due_date, total_marks):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("INSERT INTO assignment (AssignmentID, CourseID, Title, Description, DueDate, TotalMarks) VALUES (%s, %s, %s, %s, %s, %s
                   (assignment_id, course_id, title, description, due_date, total_marks))
    conn.commit()
    cursor.close()
    conn.close()

def view_assignments():
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM assignment")
    rows = cursor.fetchall()
    cursor.close()
    conn.close()
    return rows

def update_assignment(assignment_id, course_id, title, description, due_date, total_marks):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("UPDATE assignment SET CourseID=%s, Title=%s, Description=%s, DueDate=%s, TotalMarks=%s WHERE AssignmentID=%s",
                   (course_id, title, description, due_date, total_marks, assignment_id))
    conn.commit()
    cursor.close()
    conn.close()

def delete_assignment(assignment_id):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM assignment WHERE AssignmentID = %s", (assignment_id,))
    conn.commit()
    cursor.close()
    conn.close()
```

```python
# CRUD Functions for Complaint
def add_complaint(complaint_id, student_id, description, submission_date):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("INSERT INTO complaint (ComplaintID, StudentID, Description, SubmissionDate, Open, InProgress, Resolved) VALUES (%s, %s,
                   (complaint_id, student_id, description, submission_date))
    conn.commit()
    cursor.close()
    conn.close()

def view_complaints():
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM complaint")
    rows = cursor.fetchall()
    cursor.close()
    conn.close()
    return rows

def update_complaint(complaint_id, student_id, description, submission_date, open_status, in_progress, resolved):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("UPDATE complaint SET StudentID=%s, Description=%s, SubmissionDate=%s, Open=%s, InProgress=%s, Resolved=%s WHERE Complai
                   (student_id, description, submission_date, open_status, in_progress, resolved, complaint_id))
    conn.commit()
    cursor.close()
    conn.close()

def delete_complaint(complaint_id):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM complaint WHERE ComplaintID = %s", (complaint_id,))
    conn.commit()
    cursor.close()
    conn.close()
```

```python
# CRUD Functions for Enrollment
def add_enrollment(enrollment_id, course_id, student_id, progress, enrollment_date):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("INSERT INTO enrollment (EnrollmentID, CourseID, StudentID, Progress, EnrollmentDate) VALUES (%s, %s, %s, %s, %s)",
                   (enrollment_id, course_id, student_id, progress, enrollment_date))
    conn.commit()
    cursor.close()
    conn.close()

def view_enrollments():
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM enrollment")
    rows = cursor.fetchall()
    cursor.close()
    conn.close()
    return rows

def update_enrollment(enrollment_id, course_id, student_id, progress, enrollment_date):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("UPDATE enrollment SET CourseID=%s, StudentID=%s, Progress=%s, EnrollmentDate=%s WHERE EnrollmentID=%s",
                   (course_id, student_id, progress, enrollment_date, enrollment_id))
    conn.commit()
    cursor.close()
    conn.close()

def delete_enrollment(enrollment_id):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM enrollment WHERE EnrollmentID = %s", (enrollment_id,))
    conn.commit()
    cursor.close()
    conn.close()
```

```python
# CRUD Functions for Quiz
def add_quiz(quiz_id, course_id, title, total_marks):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("INSERT INTO quiz (QuizID, CourseID, Title, TotalMarks) VALUES (%s, %s, %s, %s)",
                   (quiz_id, course_id, title, total_marks))
    conn.commit()
    cursor.close()
    conn.close()

def view_quizzes():
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM quiz")
    rows = cursor.fetchall()
    cursor.close()
    conn.close()
    return rows

def update_quiz(quiz_id, course_id, title, total_marks):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("UPDATE quiz SET CourseID=%s, Title=%s, TotalMarks=%s WHERE QuizID=%s",
                   (course_id, title, total_marks, quiz_id))
    conn.commit()
    cursor.close()
    conn.close()

def delete_quiz(quiz_id):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM quiz WHERE QuizID = %s", (quiz_id,))
    conn.commit()
    cursor.close()
    conn.close()
```

```python
# CRUD Functions for Submission
def add_submission(submission_id, assignment_id, student_id, marks_obtained, submission_date):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("INSERT INTO submission (SubmissionID, AssignmentID, StudentID, MarksObtained, SubmissionDate) VALUES (%s, %s, %s, %s, %
                   (submission_id, assignment_id, student_id, marks_obtained, submission_date))
    conn.commit()
    cursor.close()
    conn.close()

def view_submissions():
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM submission")
    rows = cursor.fetchall()
    cursor.close()
    conn.close()
    return rows

def update_submission(submission_id, assignment_id, student_id, marks_obtained, submission_date):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("UPDATE submission SET AssignmentID=%s, StudentID=%s, MarksObtained=%s, SubmissionDate=%s WHERE SubmissionID=%s",
                   (assignment_id, student_id, marks_obtained, submission_date, submission_id))
    conn.commit()
    cursor.close()
    conn.close()

def delete_submission(submission_id):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM submission WHERE SubmissionID = %s", (submission_id,))
    conn.commit()
    cursor.close()
    conn.close()
```

```python
# CRUD Functions for Phone
def add_phone(student_id, phone):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("INSERT INTO phone (StudentID, Phone) VALUES (%s, %s)",
                   (student_id, phone))
    conn.commit()
    cursor.close()
    conn.close()

def view_phones():
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM phone")
    rows = cursor.fetchall()
    cursor.close()
    conn.close()
    return rows

def update_phone(student_id, phone):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("UPDATE phone SET Phone=%s WHERE StudentID=%s",
                   (phone, student_id))
    conn.commit()
    cursor.close()
    conn.close()

def delete_phone(student_id):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM phone WHERE StudentID = %s", (student_id,))
    conn.commit()
    cursor.close()
    conn.close()
```

QUERIES (JOIN QUERY, AGGREGATE FUNCTION QUERIES AND NESTED QUERY)

```
# Function to display student details with their enrolled courses (Join Query)
def show_student_enrollments():
    conn = create_connection()
    cursor = conn.cursor()
    query = """
    SELECT student.StudentID, student.Fname, student.Lname, course.Title
    FROM student
    JOIN enrollment ON student.StudentID = enrollment.StudentID
    JOIN course ON enrollment.CourseID = course.CourseID;
    """
    cursor.execute(query)
    result = cursor.fetchall()
    conn.close()
    return result
```

```
# Function to get the highest marks using SQL function
def highest_marks_in_assignment_via_function(assignment_id):
    conn = create_connection()
    cursor = conn.cursor()
    query = "SELECT get_highest_marks(%s);"
    cursor.execute(query, (assignment_id,))
    result = cursor.fetchone()
    conn.close()
    return result[0] if result else None

# Function to get the average marks using SQL function
def average_marks_in_course_via_function(course_id):
    conn = create_connection()
    cursor = conn.cursor()
    query = "SELECT get_average_marks(%s);"
    cursor.execute(query, (course_id,))
    result = cursor.fetchone()
    conn.close()
    return result[0] if result else None
```

```
query = """
    SELECT student.StudentID, student.Fname, student.Lname, course.Title
    FROM student
    JOIN enrollment ON student.StudentID = enrollment.StudentID
    JOIN course ON enrollment.CourseID = course.CourseID;
    """
```

STORED PROCEDURE, FUNCTIONS AND TRIGGERS

```
add_student_procedure | ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVIS
    IN p_student_id INT,
    IN p_email VARCHAR(255),
    IN p_dob DATE,
    IN p_fname VARCHAR(100),
    IN p_lname VARCHAR(100),
    IN p_enrollment_date DATE
)
BEGIN
    INSERT INTO student (StudentID, Email, DOB, Fname, Lname, EnrollmentDate)
    VALUES (p_student_id, p_email, p_dob, p_fname, p_lname, p_enrollment_date);
END
```

```
get_highest_marks | ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_
    DETERMINISTIC
BEGIN
    DECLARE highest_marks INT;
    SELECT MAX(MarksObtained)
    INTO highest_marks
    FROM submission
    WHERE AssignmentID = assignment_id;
    RETURN highest_marks;
END
```

```
get_average_marks | ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_
    DETERMINISTIC
BEGIN
    DECLARE avg_marks DECIMAL(10,2);
    SELECT AVG(MarksObtained)
    INTO avg_marks
    FROM submission
    JOIN assignment ON submission.AssignmentID = assignment.AssignmentID
    WHERE assignment.CourseID = course_id;
    RETURN avg_marks;
END
```

```
prevent_marks_update | UPDATE | submission | BEGIN

    IF NEW.MarksObtained > 100 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Marks cannot exceed 100!';
    END IF;
END
```

FRONT END DEVELOPMENT (FUNCTIONALITIES/FEATURES OF THE APPLICATION)

For the Smart E-Learning Platform project, the frontend has been developed using Streamlit, a Python-based framework that allows building interactive web applications efficiently. The frontend serves as the user interface through which students, instructors, and administrators can interact with the database, enabling seamless access to various functionalities. Below is an overview of the key features and functionalities provided by the application:

 # 1. User Interface and Authentication

- Login and Registration Pages: A secure login and registration page is created to authenticate users based on their role—Student, Instructor, or Administrator.
- Role-Based Access: The application dynamically adjusts the functionalities available to the user based on their role, displaying options and actions relevant to each type.

 # 2. Dashboard

- Overview Section: Provides a snapshot of the user's ongoing activities, such as courses enrolled (for students), courses taught (for instructors), and platform statistics (for administrators).
- Notifications and Updates: Displays important notifications, including upcoming deadlines for assignments, scheduled quizzes, and recent activity.

 # 3. Course Management (CRUD for Course Table)

- View Courses: Allows students to browse available courses, view course details, and see the instructor's information.
- Enroll/Unenroll from Courses: Students can self-enroll or unenroll from courses as needed, with enrollment data being automatically updated in the database.

- Course Creation/Modification: Instructors can create new courses or modify existing ones, managing details like course description and prerequisites.

 # 4. Assignment and Quiz Management (CRUD for Assignment and Quiz Tables)

- Assignment Creation and Submission: Instructors can create assignments, set deadlines, and upload assignment materials. Students can view assignment details and submit their work.
- Grading and Feedback: Instructors can grade student submissions, provide feedback, and record grades within the platform.
- Quiz Administration: Instructors can create, update, and delete quizzes for their courses. Quizzes are accessible to students based on the course they are enrolled in.

 # 5. Submission Management (CRUD for Submission Table)

- Upload Submissions: Students can upload their assignments, and each submission is timestamped to track punctuality.
- View Grades: Students can view grades and feedback for each submission once the instructor has evaluated it.

 # 6. Complaint Handling (CRUD for Complaint Table)

- File a Complaint: Students can submit complaints or concerns related to a course or instructor. This feature logs the complaint and links it to the specific course and student.
- View and Respond to Complaints: Instructors or administrators can review and respond to complaints, offering resolutions or requesting additional information from students.

 # 7. Student and Instructor Profiles (CRUD for Student and Instructor Tables)

- Profile Management: Both students and instructors can view and update their profiles, including personal details, contact information, and academic information.

- Contact Information: For instructors, the platform allows them to display contact information to students for easy communication.

 # 8. Additional Features

- Search and Filter Options: Users can search for specific courses, assignments, or students, and apply filters based on criteria such as course category, instructor, and deadline.
- Interactive Data Visualization: For administrators, the platform provides data visualizations (e.g., enrollment trends, average grades per course) to help monitor platform performance and user engagement.

 # 9. Real-Time Updates

- Automatic Data Refresh: The application updates data in real-time, ensuring that students, instructors, and administrators see the latest information without needing to refresh the page.

These features make the Smart E-Learning Platform accessible, intuitive, and efficient for users across all roles. The interface simplifies complex educational processes, facilitating smooth management of courses, assignments, and complaints.

REFERENCES/BIBLIOGRAPHY

1. MySQL Documentation

MySQL. (2024). *MySQL 8.0 Reference Manual*. Retrieved from [https://dev.mysql.com/doc/] (https://dev.mysql.com/doc/)

2. Streamlit Documentation

Streamlit. (2024). *Streamlit Documentation*. Retrieved from [https://docs.streamlit.io/] (https://docs.streamlit.io/)

3. Draw.io (Diagrams.net)

Diagrams.net. (2024). *Draw.io for ER Diagrams*. Retrieved from [https://app.diagrams.net/] (https://app.diagrams.net/)

4. Python Programming Language

Python Software Foundation. (2024). *Python 3.10 Documentation*. Retrieved from [https://docs.python.org/3/] (https://docs.python.org/3/)

5. SQL Tutorial for Database Management

W3Schools. (2024). *SQL Tutorial*. Retrieved from [https://www.w3schools.com/sql/] (https://www.w3schools.com/sql/)

6. Streamlit GitHub Repository

Streamlit GitHub. (2024). *Streamlit Examples and Use Cases*. Retrieved from [https://github.com/streamlit/streamlit] (https://github.com/streamlit/streamlit)

APPENDIX A DEFINITIONS, ACRONYMS AND ABBREVIATIONS

Definitions

- Database Management System (DBMS): A software system that enables users to define, create, maintain, and control access to databases.

- Entity-Relationship (ER) Model: A data modeling technique used to visually describe the structure of a database by identifying entities, their relationships, and their attributes.

- CRUD Operations: Acronym for Create, Read, Update, and Delete—basic operations for database management.

- Foreign Key: A field in one table that uniquely identifies a row in another table, establishing a relationship between the tables.

- Primary Key: A unique identifier for a record in a database table.

- Relational Database: A type of database that stores and provides access to data points that are related to each other.

- Stored Procedure: A saved collection of SQL statements that can be reused to perform a specific database operation.

Acronyms

- DDL: Data Definition Language

- DML: Data Manipulation Language

- DBMS: Database Management System

- ER: Entity-Relationship

- SQL: Structured Query Language

- CRUD: Create, Read, Update, Delete

- ID: Identifier

- PK: Primary Key

- FK: Foreign Key

- LMS: Learning Management System

- UI: User Interface

- API: Application Programming Interface

- VM: Virtual Machine

Abbreviations

- app: Application

- doc: Document

- ref: Reference

- mod: Module

- fn: Function

- tbl: Table