

Rakshitha Devi J
superset id: 5369940
Saveetha engineering college

Student Information System (SIS)
Implementation of OOPs

Database:

```
create database sis_db1;  
use sis_db1;
```

Table:

```
create table Student (  
    StudentID int auto_increment primary key,  
    FirstName varchar(50) not null,  
    LastName varchar(50) not null,  
    DateOfBirth date not null,  
    Email varchar(100) unique not null,  
    PhoneNumber varchar(15) unique not null  
);
```

```
create table Course (  
    CourseID int auto_increment primary key,  
    CourseName varchar(100) not null,
```

```
CourseCode varchar(20) unique not null,  
InstructorName varchar(100) not null  
);
```

```
create table Enrollment (  
    EnrollmentID int auto_increment primary key,  
    StudentID int not null,  
    CourseID int not null,  
    EnrollmentDate date not null,  
  
    foreign key (StudentID) references student(StudentID) on delete  
    cascade,  
  
    foreign key (CourseID) references course(CourseID) on delete  
    cascade,  
  
    unique (StudentID, CourseID) -- prevent duplicate enrollments  
);
```

```
create table Teacher (  
    TeacherID int auto_increment primary key,  
    FirstName varchar(50) not null,  
    LastName varchar(50) not null,  
    Email varchar(100) unique not null
```

);

create table Payment (

PaymentID int auto_increment primary key,

StudentID int not null,

Amount decimal(10,2) not null,

PaymentDate date not null,

foreign key (StudentID) references student(StudentID) on delete
cascade

);

show tables;

Code:

main.py:

```
try:
    from services.sis import SIS
except ModuleNotFoundError:
    print("Error: Could not find 'sis.py'. Ensure it exists in the correct
directory.")

def main():
    sis = SIS()

    while True:
        print("\nStudent Information System")
        print("1. Add Student")
        print("2. Enroll Student")
        print("3. Add Course")
        print("4. Add Teacher and Assign to Course")
        print("5. Assign Teacher")
        print("6. Record Payment")
        print("7. View Student Enrollments")
        print("8. View Payments by Student")
        print("9. Generate Enrollment Report")
        print("10. Exit")
        print("11. Update Student Information")
        print("12. Update Course Information")
        print("13. Update Teacher Information")

        choice = input("Enter choice: ")

        if choice == '1':
            first_name = input("First Name: ")
            last_name = input("Last Name: ")
            dob = input("Date of Birth (YYYY-MM-DD): ")
            email = input("Email: ")
            phone = input("Phone Number: ")
            sis.add_student(first_name, last_name, dob, email, phone)

        elif choice == '2':
            student_id = int(input("Enter Student ID: "))
            course_id = int(input("Enter Course ID: "))
            sis.enroll_student(student_id, course_id)

        elif choice == '3':
            course_name = input("Course Name: ")
            course_code = input("Course Code: ")
            instructor_name = input("Instructor Name: ")
            sis.add_course(course_name, course_code, instructor_name)
```

```

elif choice == "4":
    first_name = input("Enter teacher's first name: ")
    last_name = input("Enter teacher's last name: ")
    email = input("Enter teacher's email: ")
    course_id = int(input("Enter course ID to assign teacher: "))
    sis.add_teacher_and_assign_course(first_name, last_name, email,
course_id)

elif choice == '5':
    course_id = int(input("Enter Course ID: "))
    teacher_name = input("Enter Teacher Name: ")
    sis.assign_teacher(course_id, teacher_name)

elif choice == '6':
    student_id = int(input("Enter Student ID: "))
    amount = float(input("Enter Payment Amount: "))
    sis.record_payment(student_id, amount)

elif choice == '7':
    student_id = int(input("Enter Student ID: "))
    sis.view_student_enrollments(student_id)

elif choice == '8':
    student_id = int(input("Enter Student ID: "))
    sis.view_payments_by_student(student_id)

elif choice == '9':
    course_name = input("Enter Course Name for Report: ")
    sis.generate_enrollment_report(course_name)

elif choice == '10':
    print("Exiting the Student Information System. Goodbye!")
    break

elif choice == '11':
    student_id = int(input("Enter Student ID: "))
    first_name = input("Updated First Name: ")
    last_name = input("Updated Last Name: ")
    dob = input("Updated Date of Birth (YYYY-MM-DD): ")
    email = input("Updated Email: ")
    phone = input("Updated Phone Number: ")
    sis.update_student_info(student_id, first_name, last_name, dob,
email, phone)

elif choice == '12':
    course_id = int(input("Enter Course ID: "))
    course_code = input("Updated Course Code: ")
    course_name = input("Updated Course Name: ")

```

```

        instructor_name = input("Updated Instructor Name: ")
        sis.update_course_info(course_id, course_code, course_name,
instructor_name)

    elif choice == '13':
        teacher_id = int(input("Enter Teacher ID: "))
        first_name = input("Updated First Name: ")
        last_name = input("Updated Last Name: ")
        email = input("Updated Email: ")
        sis.update_teacher_info(teacher_id, first_name, last_name, email)

    else:
        print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()

```

Models:

course.py:

```

class Course:
    def __init__(self, course_id, course_name, course_code, instructor_name):
        self.__course_id = course_id
        self.__course_name = course_name
        self.__course_code = course_code
        self.__instructor_name = instructor_name
        self.__students_enrolled = []
        self.__teacher = None

    def assign_teacher(self, teacher):
        self.__teacher = teacher

    def update_course_info(self, course_code, course_name, instructor_name):
        self.__course_code = course_code
        self.__course_name = course_name
        self.__instructor_name = instructor_name

    def display_course_info(self):
        print(f"Course: {self.__course_name} ({self.__course_code}),
Instructor: {self.__instructor_name}")

    def get_enrollments(self):
        return self.__students_enrolled

    def get_teacher(self):
        return self.__teacher

    def get_course_id(self):

```

```

        return self.__course_id

    def get_course_name(self):
        return self.__course_name

    def get_course_code(self):
        return self.__course_code

    def get_instructor_name(self):
        return self.__instructor_name

```

Enrollment.py:

```

class Enrollment:
    def __init__(self, enrollment_id, student, course, enrollment_date):
        self.__enrollment_id = enrollment_id
        self.__student = student
        self.__course = course
        self.__enrollment_date = enrollment_date

        student.enrollments.append(self)
        course.enrollments.append(self)

    def get_student(self):
        return self.__student

    def get_course(self):
        return self.__course

```

Payments.py:

```

class Payment:
    def __init__(self, payment_id, student, amount, payment_date):
        self.__payment_id = payment_id
        self.__student = student
        self.__amount = amount
        self.__payment_date = payment_date

        student.get_payments().append(self)

    def get_payment_id(self):
        return self.__payment_id

    def get_student(self):
        return self.__student

    def get_payment_amount(self):
        return self.__amount

    def get_payment_date(self):
        return self.__payment_date

```

Student.py:

```
class Student:
    def __init__(self, student_id, first_name, last_name, dob, email, phone):
        self.__student_id = student_id
        self.__first_name = first_name
        self.__last_name = last_name
        self.__dob = dob
        self.__email = email
        self.__phone = phone
        self.__payments = []

    def get_student_id(self):
        return self.__student_id

    def get_first_name(self):
        return self.__first_name

    def get_last_name(self):
        return self.__last_name

    def get_date_of_birth(self):
        return self.__dob

    def get_email(self):
        return self.__email

    def get_phone_no(self):
        return self.__phone

    def get_payments(self):
        return self.__payments

    def set_first_name(self, first_name):
        self.__first_name = first_name

    def set_last_name(self, last_name):
        self.__last_name = last_name

    def set_date_of_birth(self, dob):
        self.__dob = dob

    def set_email(self, email):
        self.__email = email

    def set_phone_no(self, phone):
        self.__phone = phone
```


Teacher.py:

```
class Teacher:
    def __init__(self, teacher_id, first_name, last_name, email):
        self.__teacher_id = teacher_id
        self.__first_name = first_name
        self.__last_name = last_name
        self.__email = email
        self.__assigned_courses = []

    def update_teacher_info(self, first_name, last_name, email):
        self.__first_name = first_name
        self.__last_name = last_name
        self.__email = email

    def display_teacher_info(self):
        print(f"Teacher: {self.__first_name} {self.__last_name}, Email: {self.__email}")

    def get_assigned_courses(self):
        return self.__assigned_courses

    def get_teacher_id(self):
        return self.__teacher_id

    def get_first_name(self):
        return self.__first_name

    def get_last_name(self):
        return self.__last_name

    def get_email(self):
        return self.__email
```

Dao:

Course_dao:

```
from dao.db_connection import DBConnection
from models.course import Course

class CourseDAO:
    def __init__(self):
        self.db = DBConnection()

    def add_course(self, course):
        query = "INSERT INTO Course (CourseName, CourseCode, InstructorName) VALUES (%s, %s, %s)"
```

```

        values = (course.get_course_name(), course.get_course_code(),
course.get_instructor_name())

        with self.db.get_connection() as connection:
            with connection.cursor() as cursor:
                cursor.execute(query, values)
                connection.commit()

    def get_course_by_id(self, course_id):
        query = "SELECT CourseID, CourseName, CourseCode, InstructorName FROM
Course WHERE CourseID = %s"

        with self.db.get_connection() as connection:
            with connection.cursor() as cursor:
                cursor.execute(query, (course_id,))
                row = cursor.fetchone()

        return Course(*row) if row else None

    def assign_teacher(self, course_id, teacher_name):
        query = "UPDATE Course SET InstructorName = %s WHERE CourseID = %s"
        values = (teacher_name, course_id)

        with self.db.get_connection() as connection:
            with connection.cursor() as cursor:
                cursor.execute(query, values)
                connection.commit()

        print(f"Teacher {teacher_name} assigned to Course ID {course_id}.")

    def update_course_info(self, course_id, course_code, course_name,
instructor_name):
        query = """
UPDATE Course
SET CourseCode = %s, CourseName = %s, InstructorName = %s
WHERE CourseID = %s
"""
        values = (course_code, course_name, instructor_name, course_id)

        with self.db.get_connection() as connection:
            with connection.cursor() as cursor:
                cursor.execute(query, values)
                connection.commit()

        print(f"Course {course_id} information updated successfully.")

```

Enrollment_dao:

```
from dao.db_connection import DBConnection

class EnrollmentDAO:
    def __init__(self):
        self.db = DBConnection()

    def enroll_student(self, student_id, course_id, enrollment_date):
        try:
            with self.db.get_connection() as connection:
                with connection.cursor() as cursor:

                    check_query = """
                        SELECT COUNT(*) FROM Enrollment
                        WHERE StudentID = %s AND CourseID = %s
                    """
                    cursor.execute(check_query, (student_id, course_id))
                    (count,) = cursor.fetchone()

                    if count > 0:
                        print(f"❌ Error: Student {student_id} is already
enrolled in Course {course_id}.")
                        return

                    query = "INSERT INTO Enrollment (StudentID, CourseID,
EnrollmentDate) VALUES (%s, %s, %s)"
                    values = (student_id, course_id, enrollment_date)
                    cursor.execute(query, values)
                    connection.commit()
                    print(f"✅ Student enrolled successfully!")

        except Exception as e:
            print(f"❌ Error enrolling student: {e}")

    def get_enrollments_for_student(self, student_id):
        query = """
            SELECT c.CourseName
            FROM Enrollment e
            JOIN Course c ON e.CourseID = c.CourseID
            WHERE e.StudentID = %s
        """

        with self.db.get_connection() as connection:
            with connection.cursor() as cursor:
                cursor.execute(query, (student_id,))
                rows = cursor.fetchall()

        return [row[0] for row in rows]
```

```

def get_students_by_course(self, course_id):
    query = """
    SELECT s.StudentID, s.FirstName, s.LastName, s.Email
    FROM Enrollment e
    JOIN Student s ON e.StudentID = s.StudentID
    WHERE e.CourseID = %s
    """

    with self.db.get_connection() as connection:
        with connection.cursor() as cursor:
            cursor.execute(query, (course_id,))
            rows = cursor.fetchall()

    return [{"Student ID": row[0], "Name": f"{row[1]} {row[2]}", "Email":
row[3]} for row in rows]

def get_enrollments_by_course(self, course_name):
    query = """
    SELECT s.StudentID, s.FirstName, s.LastName, s.Email
    FROM Enrollment e
    JOIN Student s ON e.StudentID = s.StudentID
    JOIN Course c ON e.CourseID = c.CourseID
    WHERE c.CourseName = %s
    """

    with self.db.get_connection() as connection:
        with connection.cursor() as cursor:
            cursor.execute(query, (course_name,))
            rows = cursor.fetchall()

    return rows

```

Student_dao:

```

from dao.db_connection import DBConnection
from models.student import Student

class StudentDAO:
    def __init__(self):
        self.db = DBConnection()

    def add_student(self, student):
        query = "INSERT INTO Student (FirstName, LastName, DateOfBirth, Email,
PhoneNumber) VALUES (%s, %s, %s, %s, %s)"
        values = (student.get_first_name(), student.get_last_name(),
student.get_date_of_birth(), student.get_email(), student.get_phone_no())

        with self.db.get_connection() as connection:
            with connection.cursor() as cursor:

```

```

        cursor.execute(query, values)
        connection.commit()

    def get_student_by_id(self, student_id):
        query = "SELECT StudentID, FirstName, LastName, DateOfBirth, Email,
PhoneNumber FROM Student WHERE StudentID = %s"

        with self.db.get_connection() as connection:
            with connection.cursor() as cursor:
                cursor.execute(query, (student_id,))
                row = cursor.fetchone()

        return Student(*row) if row else None

    def update_balance(self, student_id, amount):

        query = "UPDATE Student SET Balance = Balance - %s WHERE StudentID =
%s"
        values = (amount, student_id)

        with self.db.get_connection() as connection:
            with connection.cursor() as cursor:
                cursor.execute(query, values)
                connection.commit()

    def update_student_info(self, student_id, first_name, last_name,
date_of_birth, email, phone_number):
        query = """
        UPDATE Student
        SET FirstName = %s, LastName = %s, DateOfBirth = %s, Email = %s,
PhoneNumber = %s
        WHERE StudentID = %s
        """
        values = (first_name, last_name, date_of_birth, email, phone_number,
student_id)

        with self.db.get_connection() as connection:
            with connection.cursor() as cursor:
                cursor.execute(query, values)
                connection.commit()

        print(f"Student {student_id} information updated successfully.")

```

Payment_dao:

```

from dao.db_connection import DBConnection
from models.payment import Payment
from dao.student_dao import StudentDAO # Import StudentDAO

```

```

class PaymentDAO:
    def __init__(self):
        self.db = DBConnection()
        self.student_dao = StudentDAO()

    def record_payment(self, student_id, amount, payment_date):
        query = "INSERT INTO Payment (StudentID, Amount, PaymentDate) VALUES (%s, %s, %s)"
        values = (student_id, amount, payment_date)

        with self.db.get_connection() as connection:
            with connection.cursor() as cursor:
                cursor.execute(query, values)
                connection.commit()

    def get_payments_for_student(self, student_id):
        query = "SELECT PaymentID, StudentID, Amount, PaymentDate FROM Payment WHERE StudentID = %s"

        with self.db.get_connection() as connection:
            with connection.cursor() as cursor:
                cursor.execute(query, (student_id,))
                rows = cursor.fetchall()

        payments = []
        student = self.student_dao.get_student_by_id(student_id)

        for row in rows:
            payments.append(Payment(row[0], student, row[2], row[3]))

        return payments

```

Teacher_dao:

```

from dao.db_connection import DBConnection
from models.teacher import Teacher

class TeacherDAO:
    def __init__(self):
        self.db = DBConnection()

    def add_teacher(self, teacher):
        query = "INSERT INTO Teacher (FirstName, LastName, Email) VALUES (%s, %s, %s)"
        values = (teacher.get_first_name(), teacher.get_last_name(), teacher.get_email())

        with self.db.get_connection() as connection:
            with connection.cursor() as cursor:
                cursor.execute(query, values)
                connection.commit()

```

```

        print(f"Teacher {teacher.get_first_name()} {teacher.get_last_name()}
added successfully.")

def get_teacher_by_id(self, teacher_id):
    query = "SELECT TeacherID, FirstName, LastName, Email FROM Teacher
WHERE TeacherID = %s"

    with self.db.get_connection() as connection:
        with connection.cursor() as cursor:
            cursor.execute(query, (teacher_id,))
            row = cursor.fetchone()

    return Teacher(*row) if row else None

def update_teacher_info(self, teacher_id, first_name, last_name, email):
    query = """
UPDATE Teacher
SET FirstName = %s, LastName = %s, Email = %s
WHERE TeacherID = %s
"""
    values = (first_name, last_name, email, teacher_id)

    with self.db.get_connection() as connection:
        with connection.cursor() as cursor:
            cursor.execute(query, values)
            connection.commit()

    print(f"Teacher {teacher_id} information updated successfully.")

```

DB_connection:

```

import mysql.connector

class DBConnection:
    def __init__(self):
        self.host = "localhost"
        self.user = "root"
        self.password = "rakshi430"
        self.database = "sis_db1"
        self.conn = None
        self.cursor = None

    def get_connection(self):

        return mysql.connector.connect(
            host=self.host,
            user=self.user,

```

```

        password=self.password,
        database=self.database
    )

def disconnect(self):

    if self.cursor:
        self.cursor.close()
    if self.conn:
        self.conn.close()

def execute_query(self, query, params=None):

    self.get_connection()
    self.cursor.execute(query, params or ())
    self.conn.commit()
    self.disconnect()

def fetch_results(self, query, params=None):

    self.get_connection()
    self.cursor.execute(query, params or ())
    results = self.cursor.fetchall()
    self.disconnect()
    return results

```

Exception:

exceptions.py:

```

class StudentNotFoundException(Exception):
    pass

class CourseNotFoundException(Exception):
    pass

class TeacherNotFoundException(Exception):
    pass

class DuplicateEnrollmentException(Exception):
    pass

class PaymentValidationException(Exception):
    pass

class InvalidStudentDataException(Exception):
    pass

class InvalidCourseDataException(Exception):
    pass

```



```

class InvalidEnrollmentDataException(Exception):
    pass

class InvalidTeacherDataException(Exception):
    pass

class InsufficientFundsException(Exception):
    pass

```

Services:

sis.py:

```

from dao.student_dao import StudentDAO
from dao.course_dao import CourseDAO
from dao.enrollment_dao import EnrollmentDAO
from dao.payment_dao import PaymentDAO
from dao.teacher_dao import TeacherDAO
from models.student import Student
from models.course import Course
from models.teacher import Teacher

import datetime

class SIS:
    def __init__(self):
        self.student_dao = StudentDAO()
        self.course_dao = CourseDAO()
        self.enrollment_dao = EnrollmentDAO()
        self.payment_dao = PaymentDAO()
        self.teacher_dao = TeacherDAO()

    def add_student(self, first_name, last_name, dob, email, phone):
        student = Student(None, first_name, last_name, dob, email, phone)
        self.student_dao.add_student(student)
        print("Student added successfully!")

    def enroll_student(self, student_id, course_id):
        enrollment_date = datetime.date.today()
        self.enrollment_dao.enroll_student(student_id, course_id,
enrollment_date)
        print("Student enrolled successfully!")

    def add_course(self, course_name, course_code, instructor_name):
        course = Course(None, course_name, course_code, instructor_name)
        self.course_dao.add_course(course)
        print("Course added successfully!")

```

```

def add_teacher_and_assign_course(self, first_name, last_name, email,
course_id):
    teacher = Teacher(None, first_name, last_name, email)
    self.teacher_dao.add_teacher(teacher)
    teacher_name = f"{first_name} {last_name}"
    self.course_dao.assign_teacher(course_id, teacher_name)
    print(f"Teacher {teacher_name} assigned to Course ID {course_id}.")

def assign_teacher(self, course_id, teacher_name):
    self.course_dao.assign_teacher(course_id, teacher_name)
    print("Teacher assigned successfully!")

def record_payment(self, student_id, amount):
    payment_date = datetime.date.today()
    self.payment_dao.record_payment(student_id, amount, payment_date)
    print("Payment recorded successfully!")

def view_student_enrollments(self, student_id):
    course_names =
self.enrollment_dao.get_enrollments_for_student(student_id)
    if course_names:
        print(f"Student {student_id} is enrolled in: {'',
'.join(course_names)}")
    else:
        print(f"Student {student_id} is not enrolled in any courses.")

def view_payments_by_student(self, student_id):
    payments = self.payment_dao.get_payments_for_student(student_id)
    if not payments:
        print("No payments found for this student.")
        return
    for payment in payments:
        print(f"Payment ID: {payment.get_payment_id()}, Amount:
{payment.get_payment_amount()}, Date: {payment.get_payment_date()}")

def generate_enrollment_report(self, course_name):
    students = self.enrollment_dao.get_enrollments_by_course(course_name)
    if not students:
        print(f"No enrollments found for course: {course_name}")
        return
    print(f"\nEnrollment Report for {course_name}")
    print("-" * 50)
    for student in students:
        print(f"Student ID: {student[0]}, Name: {student[1]} {student[2]},
Email: {student[3]}")
    print("-" * 50)

```

```

def update_student_info(self, student_id, first_name, last_name, dob,
email, phone):

    try:
        self.student_dao.update_student_info(student_id, first_name,
last_name, dob, email, phone)
        print(f"✅ Student (ID: {student_id}) updated successfully.")
    except Exception as e:
        print(f"❌ Error updating student: {e}")

def update_course_info(self, course_id, course_code, course_name,
instructor_name):

    try:
        self.course_dao.update_course_info(course_id, course_code,
course_name, instructor_name)
        print(f"✅ Course (ID: {course_id}) updated successfully.")
    except Exception as e:
        print(f"❌ Error updating course: {e}")

def update_teacher_info(self, teacher_id, first_name, last_name, email):

    try:
        self.teacher_dao.update_teacher_info(teacher_id, first_name,
last_name, email)
        print(f"✅ Teacher (ID: {teacher_id}) updated successfully.")
    except Exception as e:
        print(f"❌ Error updating teacher: {e}")

```

OUTPUT:

Add students:

```
Student Information System
1. Add Student
2. Enroll Student
3. Add Course
4. Add Teacher and Assign to Course
5. Assign Teacher
6. Record Payment
7. View Student Enrollments
8. View Payments by Student
9. Generate Enrollment Report
10. Exit
11. Update Student Information
12. Update Course Information
13. Update Teacher Information
Enter choice: 1
First Name: John
Last Name: Doe
Date of Birth (YYYY-MM-DD): 1995-08-15
Email: john.doe@gmail.com
Phone Number: 1234567890
Student added successfully!
```

	StudentID	FirstName	LastName	DateOfBirth	Email	PhoneN
▶	1	John	Doe	1995-08-15	john.doe@gmail.com	1234567
✱	NULL	NULL	NULL	NULL	NULL	NULL

Add course:

```
Student Information System
1. Add Student
2. Enroll Student
3. Add Course
4. Add Teacher and Assign to Course
5. Assign Teacher
6. Record Payment
7. View Student Enrollments
8. View Payments by Student
9. Generate Enrollment Report
10. Exit
11. Update Student Information
12. Update Course Information
13. Update Teacher Information
Enter choice: 3
Course Name:  Introduction to Programming
Course Code:  IP234
Instructor Name:  shobana
Course added successfully!
```

```
Student Information System
1. Add Student
2. Enroll Student
3. Add Course
4. Add Teacher and Assign to Course
5. Assign Teacher
6. Record Payment
7. View Student Enrollments
8. View Payments by Student
9. Generate Enrollment Report
10. Exit
11. Update Student Information
12. Update Course Information
13. Update Teacher Information
Enter choice: 3
Course Name:  Mathematics 101
Course Code:  MS101
Instructor Name:  shalini
Course added successfully!
```

```

Student Information System
1. Add Student
2. Enroll Student
3. Add Course
4. Add Teacher and Assign to Course
5. Assign Teacher
6. Record Payment
7. View Student Enrollments
8. View Payments by Student
9. Generate Enrollment Report
10. Exit
11. Update Student Information
12. Update Course Information
13. Update Teacher Information
Enter choice: 3
Course Name: Advanced Database Management
Course Code: CS302
Instructor Name: Sarah Smith
Course added successfully!

```

	CourseID	CourseName	CourseCode	InstructorName
▶	1	Introduction to Programming	IP234	shobana
	2	Mathematics 101	MS101	shalini
	3	Advanced Database Management	CS302	Sarah Smith
•	NULL	NULL	NULL	NULL

Assign Teacher and assign to course:

```
Student Information System
1. Add Student
2. Enroll Student
3. Add Course
4. Add Teacher and Assign to Course
5. Assign Teacher
6. Record Payment
7. View Student Enrollments
8. View Payments by Student
9. Generate Enrollment Report
10. Exit
11. Update Student Information
12. Update Course Information
13. Update Teacher Information
Enter choice: 4
Enter teacher's first name: Sarah
Enter teacher's last name: Smith
Enter teacher's email: sarah.smith@example.com
Enter course ID to assign teacher: 3
Teacher Sarah Smith added successfully.
Teacher Sarah Smith assigned to Course ID 3.
```

Enroll student:

```
Student Information System
1. Add Student
2. Enroll Student
3. Add Course
4. Add Teacher and Assign to Course
5. Assign Teacher
6. Record Payment
7. View Student Enrollments
8. View Payments by Student
9. Generate Enrollment Report
10. Exit
11. Update Student Information
12. Update Course Information
13. Update Teacher Information
Enter choice: 2
Enter Student ID: 1
Enter Course ID: 1
Student enrolled successfully!
```

	EnrollmentID	StudentID	CourseID	EnrollmentDate
▶	1	1	1	2025-04-02
•	NULL	NULL	NULL	NULL

Assign Teacher:

```
Student Information System
1. Add Student
2. Enroll Student
3. Add Course
4. Add Teacher and Assign to Course
5. Assign Teacher
6. Record Payment
7. View Student Enrollments
8. View Payments by Student
9. Generate Enrollment Report
10. Exit
11. Update Student Information
12. Update Course Information
13. Update Teacher Information
Enter choice: 5
Enter Course ID: 3
Enter Teacher Name: 1
Teacher 1 assigned to Course ID 3.
Teacher assigned successfully!
```

Record payments:

```
Student Information System
1. Add Student
2. Enroll Student
3. Add Course
4. Add Teacher and Assign to Course
5. Assign Teacher
6. Record Payment
7. View Student Enrollments
8. View Payments by Student
9. Generate Enrollment Report
10. Exit
11. Update Student Information
12. Update Course Information
13. Update Teacher Information
Enter choice: 6
Enter Student ID: 1
Enter Payment Amount: 500
Payment recorded successfully!
```

	PaymentID	StudentID	Amount	PaymentDate
▶	1	1	500.00	2025-04-02
*	NULL	NULL	NULL	NULL

View Student Enrollments:

```
Student Information System
1. Add Student
2. Enroll Student
3. Add Course
4. Add Teacher and Assign to Course
5. Assign Teacher
6. Record Payment
7. View Student Enrollments
8. View Payments by Student
9. Generate Enrollment Report
10. Exit
11. Update Student Information
12. Update Course Information
13. Update Teacher Information
Enter choice: 7
Enter Student ID: 1
Student 1 is enrolled in: Introduction to Programming
```

View Payments by Student:

```
Student Information System
1. Add Student
2. Enroll Student
3. Add Course
4. Add Teacher and Assign to Course
5. Assign Teacher
6. Record Payment
7. View Student Enrollments
8. View Payments by Student
9. Generate Enrollment Report
10. Exit
11. Update Student Information
12. Update Course Information
13. Update Teacher Information
Enter choice: 8
Enter Student ID: 1
Payment ID: 1, Amount: 500.00, Date: 2025-04-02
```

View student enrollment:

```
Student Information System
1. Add Student
2. Enroll Student
3. Add Course
4. Add Teacher and Assign to Course
5. Assign Teacher
6. Record Payment
7. View Student Enrollments
8. View Payments by Student
9. Generate Enrollment Report
10. Exit
11. Update Student Information
12. Update Course Information
13. Update Teacher Information
Enter choice: 7
Enter Student ID: 1
Student 1 is enrolled in: Introduction to Programming, Computer Science
```

Generate Enrollment Report:

```
Student Information System
1. Add Student
2. Enroll Student
3. Add Course|
4. Add Teacher and Assign to Course
5. Assign Teacher
6. Record Payment
7. View Student Enrollments
8. View Payments by Student
9. Generate Enrollment Report
10. Exit
11. Update Student Information
12. Update Course Information
13. Update Teacher Information
Enter choice: 9
Enter Course Name for Report: Computer Science

Enrollment Report for Computer Science
-----
Student ID: 1, Name: John Doe, Email: john.doe@gmail.com
-----
```

Update Student Information:

	StudentID	FirstName	LastName	DateOfBirth	Email	PhoneN
▶	1	John	Doe	1995-08-15	john.doe@gmail.com	1234567
	2	Rakshitha	Devi J	2003-08-04	rakshitha@gmail.com	9876543
•	NULL	NULL	NULL	NULL	NULL	NULL

```
Student Information System
1. Add Student
2. Enroll Student
3. Add Course
4. Add Teacher and Assign to Course
5. Assign Teacher
6. Record Payment
7. View Student Enrollments
8. View Payments by Student
9. Generate Enrollment Report
10. Exit
11. Update Student Information
12. Update Course Information
13. Update Teacher Information
Enter choice: 11
Enter Student ID: 2
Updated First Name: Madhu
Updated Last Name: shrie j
Updated Date of Birth (YYYY-MM-DD): 2006-05-07
Updated Email: madhu@gmail.com
Updated Phone Number: 1234123456
Student 2 information updated successfully.
✅ Student (ID: 2) updated successfully.
```

	StudentID	FirstName	LastName	DateOfBirth	Email	PhoneN
▶	1	John	Doe	1995-08-15	john.doe@gmail.com	1234567
	2	Rakshitha	Devi J	2003-08-04	rakshitha@gmail.com	9876543
•	NULL	NULL	NULL	NULL	NULL	NULL

Update Course Information:

	CourseID	CourseName	CourseCode	InstructorName
▶	1	Introduction to Programming	IP234	1
	2	Mathematics 101	MS101	shalini
	3	Advanced Database Management	CS302	1
	4	Computer Science	101	sarah smith
✱	NULL	NULL	NULL	NULL

```
Student Information System
1. Add Student
2. Enroll Student
3. Add Course
4. Add Teacher and Assign to Course
5. Assign Teacher
6. Record Payment
7. View Student Enrollments
8. View Payments by Student
9. Generate Enrollment Report
10. Exit
11. Update Student Information
12. Update Course Information
13. Update Teacher Information
Enter choice: 12
Enter Course ID: 1
Updated Course Code: IP234
Updated Course Name: python programming
Updated Instructor Name: sheela
Course 1 information updated successfully.
✅ Course (ID: 1) updated successfully.
```

	CourseID	CourseName	CourseCode	InstructorName
▶	1	python programming	IP234	sheela
	2	Mathematics 101	MS101	shalini
	3	Advanced Database Management	CS302	1
	4	Computer Science	101	sarah smith
✱	NULL	NULL	NULL	NULL

Update Teacher Information:

	TeacherID	FirstName	LastName	Email
▶	1	Sarah	Smith	sarah.smith@example.com
	2	sheela	s	sheela@gmail.com
*	NULL	NULL	NULL	NULL

Student Information System

1. Add Student
2. Enroll Student
3. Add Course
4. Add Teacher and Assign to Course
5. Assign Teacher
6. Record Payment
7. View Student Enrollments
8. View Payments by Student
9. Generate Enrollment Report
10. Exit
11. Update Student Information
12. Update Course Information
13. Update Teacher Information

Enter choice: 13

Enter Teacher ID: 2

Updated First Name: pooja

Updated Last Name: k

Updated Email: pooja@gmail.com

Teacher 2 information updated successfully.

✅ Teacher (ID: 2) updated successfully.

	TeacherID	FirstName	LastName	Email
▶	1	Sarah	Smith	sarah.smith@example.com
	2	pooja	k	pooja@gmail.com
*	NULL	NULL	NULL	NULL