ASSIGNMENT 2 PYTHON

NAME :- SUSHANT KUMAR SINGH

Project:- Student Information System (SIS)

Student Information System (SIS)

Implement OOPs

A Student Information System (SIS) manages information about students, courses, student enrollments, teachers, and payments. Each student can enroll in multiple courses, each course can have multiple students, each course is taught by a teacher, and students make payments for their courses. Students have attributes such as name, date of birth, email, and phone number. Courses have attributes such as course name, course code, and instructor name. Enrollments track which students are enrolled in which courses. Teachers have attributes such as names and email. Payments track the amount and date of payments made by students.

Task 1|2: Define Classes|Methods

Define the following classes based on the domain description:

Student class with the following attributes:

- Student ID
- First Name
- Last Name
- Date of Birth
- Email
- Phone Number

```python
from enrollment import Enrollment
from datetime import datetime
from payment import Payment
2 usages
class Student:
    def __init__(self, student_id, first_name, last_name, date_of_birth, email, phone_number):
        self.student_id = student_id
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number
        self.enrollments = []
        self.payments = []
    def enroll_in_course(self, course):
        enrollment_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        enrollment = Enrollment(len(self.enrollments) + 1, self, course, enrollment_date)
        self.enrollments.append(enrollment)
        course.enrollments.append(enrollment)
        print(f"{self.first_name} {self.last_name} enrolled in {course.course_name}.")
    def update_student_info(self, first_name, last_name, date_of_birth, email, phone_number):
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
```

```python
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number
        print(f"Student information updated for {self.first_name} {self.last_name}.")
    def make_payment(self, amount, payment_date):
        payment = Payment(len(self.payments) + 1, self, amount, payment_date)
        self.payments.append(payment)
        print(f"Payment of ${amount} made by {self.first_name} {self.last_name} on {payment_date}.")
    1 usage (1 dynamic)
    def display_student_info(self):
        print(f"Student ID: {self.student_id}")
        print(f"Name: {self.first_name} {self.last_name}")
        print(f"Date of Birth: {self.date_of_birth}")
        print(f"Email: {self.email}")
        print(f"Phone Number: {self.phone_number}")

    1 usage (1 dynamic)
    def get_enrolled_courses(self):
        return [enrollment.course for enrollment in self.enrollments]

    1 usage (1 dynamic)
    def get_payment_history(self):
```

Input/Output given by User

```
C:\Users\ssush\PycharmProjects\SIS5\venv\Scripts\python.exe C:\Users\ssush\PycharmProjects\SIS5\main.py

1. Add a New Student
2. Enroll Student in a Course
3. Assign Teacher to a Course
4. Make Payment
5. Display Student Information
6. Display Enrolled Courses
7. Add New Teacher
8. Display Payment History
9. Exit
Enter your choice (1-10): 1
Enter student's first name: Sushant
Enter student's last name: Kumar
Enter of birth date (YYYY-MM-DD): 2001-10-31
Enter student's email: ssushant776@gmail.com
Enter student's phone number: 96542535425
Student added successfully.
Student added successfully.
```

```
mysql> SELECT * FROM students;
+----+------------+-----------+------------+----------------------+--------------+
| id | first_name | last_name | birth_date | email                | phone        |
+----+------------+-----------+------------+----------------------+--------------+
|  1 | RAMA       | KUMAR     | 0500-01-22 | rama@gmail.com       | 9652458358   |
|  2 | aman       | singh     | 2024-09-09 | aman@gmail.com       | 9652321459   |
|  3 | jane       | jhonson   | 2024-01-30 | jane@yahoo.com       | 965243585    |
|  4 | Jhon       | Doe       | 1995-08-15 | jhon.doe@example.com | 123-456-7890 |
|  5 | RAMA       | KUMAR     | 0500-01-22 | rama@gmail.com       | 9652458358   |
|  6 | Sushant    | Kumar     | 2001-10-31 | ssushant776@gmail.com | 96542535425  |
+----+------------+-----------+------------+----------------------+--------------+
6 rows in set (0.01 sec)
```

Data successfully saved into the database

Course class with the following attributes:

- Course ID
- Course Name
- Course Code
- Instructor Name

```python
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.teacher = None
        self.enrollments = []
    def assign_teacher(self, teacher):
        self.teacher = teacher
        teacher.assigned_courses.append(self)
        print(f"{teacher.first_name} {teacher.last_name} assigned to teach {self.course_name}.")
    def update_course_info(self, course_code, course_name, instructor):
        self.course_code = course_code
        self.course_name = course_name
        self.instructor_name = instructor
        print(f"Course information updated for {self.course_name}.")
    def display_course_info(self):
        print(f"Course ID: {self.course_id}")
        print(f"Course Name: {self.course_name}")
        print(f"Course Code: {self.course_code}")
        print(f"Instructor: {self.instructor_name}")
    def get_enrollments(self):
        return self.enrollments
    def get_teacher(self):
        return self.teacher
```

Enrollment class to represent the relationship between students and courses. It should have attributes:

- Enrollment ID
- Student ID (reference to a Student)
- Course ID (reference to a Course)
- Enrollment Date

```python
from datetime import datetime
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

    def get_student(self):
        return self.student

    def get_course(self):
        return self.course
```

Teacher class with the following attributes:

- Teacher ID
- First Name
- Last Name
- Email

```python
from datetime import datetime
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 = []

    1 usage (1 dynamic)
    def update_teacher_info(self, name, email, expertise):
        self.first_name = name
        self.email = email
        print(f"Teacher information updated for {self.first_name} {self.last_name}.")

    def display_teacher_info(self):
        print(f"Teacher ID: {self.teacher_id}")
        print(f"Name: {self.first_name} {self.last_name}")
        print(f"Email: {self.email}")

    def get_assigned_courses(self):
        self.assigned_courses
        print(f"assigned_courses{self.assigned_courses}.")
```

Payment class with the following attributes:

- Payment ID
- Student ID (reference to a Student)
- Amount
- Payment Date

```python
from datetime import datetime
4 usages
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

    def get_student(self):
        return self.student

    1 usage (1 dynamic)
    def get_payment_amount(self):
        return self.amount

    1 usage (1 dynamic)
    def get_payment_date(self):
        return self.payment_date
```

Task 3: Implement Methods

Implement methods in your classes to perform various operations related to the Student Information System (SIS). These methods will allow you to interact with and manipulate data within your system.

Below are detailed instructions on how to implement methods in each class:

Implement the following methods in the appropriate classes:

Student Class:

• EnrollInCourse(course: Course): Enrolls the student in a course.
• UpdateStudentInfo(firstName: string, lastName: string, dateOfBirth: DateTime, email: string, phoneNumber: string): Updates the student's information.
• MakePayment(amount: decimal, paymentDate: DateTime): Records a payment made by the student.
• DisplayStudentInfo(): Displays detailed information about the student.
• GetEnrolledCourses(): Retrieves a list of courses in which the student is enrolled.
• GetPaymentHistory(): Retrieves a list of payment records for the student.

```python
from enrollment import Enrollment
from datetime import datetime
from payment import Payment
2 usages
class Student:
    def __init__(self, student_id, first_name, last_name, date_of_birth, email, phone_number):
        self.student_id = student_id
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number
        self.enrollments = []
        self.payments = []
    def enroll_in_course(self, course):
        enrollment_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        enrollment = Enrollment(len(self.enrollments) + 1, self, course, enrollment_date)
        self.enrollments.append(enrollment)
        course.enrollments.append(enrollment)
        print(f"{self.first_name} {self.last_name} enrolled in {course.course_name}.")
    def update_student_info(self, first_name, last_name, date_of_birth, email, phone_number):
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
```

```python
    def update_student_info(self, first_name, last_name, date_of_birth, email, phone_number):
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number
        print(f"Student information updated for {self.first_name} {self.last_name}.")
    def make_payment(self, amount, payment_date):
        payment = Payment(len(self.payments) + 1, self, amount, payment_date)
        self.payments.append(payment)
        print(f"Payment of ${amount} made by {self.first_name} {self.last_name} on {payment_date}.")
    1 usage (1 dynamic)
    def display_student_info(self):
        print(f"Student ID: {self.student_id}")
        print(f"Name: {self.first_name} {self.last_name}")
        print(f"Date of Birth: {self.date_of_birth}")
        print(f"Email: {self.email}")
        print(f"Phone Number: {self.phone_number}")

    1 usage (1 dynamic)
    def get_enrolled_courses(self):
        return [enrollment.course for enrollment in self.enrollments]

    1 usage (1 dynamic)
    def get_payment_history(self):
        return self.payments
```

Course Class:

• AssignTeacher(teacher: Teacher): Assigns a teacher to the course.

• UpdateCourseInfo(courseCode: string, courseName: string, instructor: string): Updates course information.

• DisplayCourseInfo(): Displays detailed information about the course.

• GetEnrollments(): Retrieves a list of student enrollments for the course.

• GetTeacher(): Retrieves the assigned teacher for the course.

```python
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.teacher = None
        self.enrollments = []
    def assign_teacher(self, teacher):
        self.teacher = teacher
        teacher.assigned_courses.append(self)
        print(f"{teacher.first_name} {teacher.last_name} assigned to teach {self.course_name}.")
    def update_course_info(self, course_code, course_name, instructor):
        self.course_code = course_code
        self.course_name = course_name
        self.instructor_name = instructor
        print(f"Course information updated for {self.course_name}.")
    def display_course_info(self):
        print(f"Course ID: {self.course_id}")
        print(f"Course Name: {self.course_name}")
        print(f"Course Code: {self.course_code}")
        print(f"Instructor: {self.instructor_name}")
    def get_enrollments(self):
        return self.enrollments
    def get_teacher(self):
        return self.teacher
```

Enrollment Class:

• GetStudent(): Retrieves the student associated with the enrollment.

• GetCourse(): Retrieves the course associated with the enrollment.

```python
from datetime import datetime
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

    def get_student(self):
        return self.student

    def get_course(self):
        return self.course
```

Teacher Class:

• UpdateTeacherInfo(name: string, email: string, expertise: string): Updates teacher information.

• DisplayTeacherInfo(): Displays detailed information about the teacher.

• GetAssignedCourses(): Retrieves a list of courses assigned to the teacher.

```python
from datetime import datetime
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 = []

    1 usage (1 dynamic)
    def update_teacher_info(self, name, email, expertise):
        self.first_name = name
        self.email = email
        print(f"Teacher information updated for {self.first_name} {self.last_name}.")

    def display_teacher_info(self):
        print(f"Teacher ID: {self.teacher_id}")
        print(f"Name: {self.first_name} {self.last_name}")
        print(f"Email: {self.email}")

    def get_assigned_courses(self):
        self.assigned_courses
        print(f"assigned_courses{self.assigned_courses}.")
```

Payment Class:

• GetStudent(): Retrieves the student associated with the payment.

• GetPaymentAmount(): Retrieves the payment amount.

• GetPaymentDate(): Retrieves the payment date.

main.py | DatabaseConnector.py | student.py | course.py | enrollment.py | teacher.py | payme

ojects\SIS5

```python
from datetime import datetime
4 usages
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

    def get_student(self):
        return self.student

    1 usage (1 dynamic)
    def get_payment_amount(self):
        return self.amount

    1 usage (1 dynamic)
    def get_payment_date(self):
        return self.payment_date
```

Use the Methods

In your driver program or any part of your code where you want to perform actions related to the Student Information System, create instances of your classes, and use the methods you've implemented.

Repeat this process for using other methods you've implemented in your classes and the SIS class.

Task 4: Exceptions handling and Custom Exceptions

Implementing custom exceptions allows you to define and throw exceptions tailored to specific situations or business logic requirements.

• **DuplicateEnrollmentException**: Thrown when a student is already enrolled in a course and tries to enroll again. This exception can be used in the EnrollStudentInCourse method.

xception.py | InvalidTeacherDataException.py | InsufficientFundsException.py | student.py | DuplicateEnrollmentException.py

```python
2 usages
class DuplicateEnrollmentException(Exception):
    def __init__(self, student_id, course_id):
        super().__init__(f"Student with ID {student_id} is already enrolled in course with ID {course_id}.")

```

```python
from datetime import datetime
from enrollment import Enrollment
from payment import Payment
from DuplicateEnrollmentException import DuplicateEnrollmentException


2 usages
class Student:
    def __init__(self, student_id, first_name, last_name, date_of_birth, email, phone_number):
        self.student_id = student_id
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number
        self.enrollments = []
        self.payments = []

    def enroll_in_course(self, course):
        if self.is_student_already_enrolled(course):
            raise DuplicateEnrollmentException(self.student_id, course.course_id)

        enrollment_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        enrollment = Enrollment(len(self.enrollments) + 1, self, course, enrollment_date)
        self.enrollments.append(enrollment)
        course.enrollments.append(enrollment)
        print(f"{self.first_name} {self.last_name} enrolled in {course.course_name}.")
```

StudentNotFoundException:

aException.py | InvalidTeacherDataException.py | InsufficientFundsException.py | student.py | StudentNotFoundExcept

```python
class StudentNotFoundException(Exception):
    def __init__(self, student_id):
        super().__init__(f"Student with ID {student_id} not found.")

```

TeacherNotFoundException:

aException.py | InsufficientFundsException.py | student.py | StudentNotFoundException.py | TeacherNotFoundExce

```python
class TeacherNotFoundException(Exception):
    def __init__(self, teacher_id):
        super().__init__(f"Teacher with ID {teacher_id} not found.")
```

PaymentValidationException:

```python
class PaymentValidationException(Exception):
    def __init__(self, message):
        super().__init__(message)
```

InvalidStudentDataException:

```python
class PaymentValidationException(Exception):
    def __init__(self, message):
        super().__init__(message)
```

InvalidCourseDataException:

```python
class InvalidCourseDataException(Exception):
    def __init__(self, message):
        super().__init__(message)
```

InvalidEnrollmentDataException:

```python
class InvalidEnrollmentDataException(Exception):
    def __init__(self, message):
        super().__init__(message)
```

InvalidTeacherDataException:

```python
class InvalidTeacherDataException(Exception):
    def __init__(self, message):
        super().__init__(message)
```

Task 5: Collections

Implement Collections:
Implement relationships between classes using appropriate data structures (e.g., lists or **dictionaries**) to maintain associations between students, courses, enrollments, teachers, and payments.
These relationships are essential for the Student Information System (SIS) to track and manage student enrollments, teacher assignments, and payments accurately.

Student Class: Create a list or collection property to store the student's enrollments. This property will hold references to Enrollment objects.

```
1 usage
class Student:
    def __init__(self, student_id, first_name, last_name, birth_date, email, phone):
        self._student_id = student_id
        self._first_name = first_name
        self._last_name = last_name
        self._birth_date = birth_date
        self._email = email
        self._phone = phone
        self._enrollments = []  # List to store Enrollment objects

    1 usage (1 dynamic)
    @property
    def student_id(self):
        return self._student_id

    2 usages (2 dynamic)
    @property
    def first_name(self):
        return self._first_name

    2 usages (2 dynamic)
    @property
    def last_name(self):
```

```
    @property
    def birth_date(self):
        return self._birth_date

    @property
    def email(self):
        return self._email

    @property
    def phone(self):
        return self._phone

    3 usages (1 dynamic)
    @property
    def enrollments(self):
        return self._enrollments

    2 usages (1 dynamic)
    @enrollments.setter
    def enrollments(self, enrollment):
        self._enrollments.append(enrollment)
```

Course Class:
Create a list or collection property to store the course's enrollments. This property will hold references to Enrollment objects.

```python
1 usage
class Course:
    def __init__(self, course_code, course_name):
        self._course_code = course_code
        self._course_name = course_name
        self._enrollments = []

    4 usages (1 dynamic)
    @property
    def enrollments(self):
        return self._enrollments

    2 usages (1 dynamic)
    @enrollments.setter
    def enrollments(self, enrollment):
        self._enrollments.append(enrollment)

    1 usage (1 dynamic)
    @enrollments.deleter
    def enrollments(self):
        self._enrollments.clear()
```

Enrollment Class:
Include properties to hold references to both the Student and Course objects.

course.py enrollment.py ENROLLMENTT.PY StudentNotFoundException.py TeacherNotFoundException.py

```python
1 usage
class Enrollment:
    def __init__(self, student, course):
        self._student = student
        self._course = course

    2 usages
    @property
    def student(self):
        return self._student

    @student.setter
    def student(self, student):
        self._student = student

    3 usages (1 dynamic)
    @property
    def course(self):
        return self._course

    1 usage (1 dynamic)
    @course.setter
    def course(self, course):
        self._course = course
```

Payment Class: Include a property to hold a reference to the Student object. Example: Student Student { get; set; }

```python
1 usage
class Payment:
    def __init__(self, payment_id, amount, payment_date, student):
        self._payment_id = payment_id
        self._amount = amount
        self._payment_date = payment_date
        self._student = student

    @property
    def student(self):
        return self._student

    @student.setter
    def student(self, new_student):
        self._student = new_student

    @student.deleter
    def student(self):
        del self._student
```

Teacher Class:
Create a list or collection property to store the teacher's assigned courses. This property will hold references to Course objects.

```python
1 usage
class Teacher:
    def __init__(self, teacher_id, name, email, assigned_courses=None):
        self._teacher_id = teacher_id
        self._name = name
        self._email = email
        self._assigned_courses = assigned_courses or []

    4 usages (1 dynamic)
    @property
    def assigned_courses(self):
        return self._assigned_courses

    1 usage (1 dynamic)
    @assigned_courses.setter
    def assigned_courses(self, new_courses):
        self._assigned_courses = new_courses

    1 usage (1 dynamic)
    @assigned_courses.deleter
    def assigned_courses(self):
        del self._assigned_courses
```

Task 6: Create Methods for Managing Relationships

AddEnrollment(student, course, enrollmentDate): In the SIS class, create a method that adds an enrollment to both the Student's and Course's enrollment lists. Ensure the Enrollment object references the correct Student and Course.

- AssignCourseToTeacher(course, teacher): In the SIS class, create a method to assign a course to a teacher. Add the course to the teacher's AssignedCourses list
- AddPayment(student, amount, paymentDate): In the SIS class, create a method that adds a payment to the Student's payment history. Ensure the Payment object references the correct Student.
- GetEnrollmentsForStudent(student): In the SIS class, create a method to retrieve all enrollments for a specific student.
- GetCoursesForTeacher(teacher): In the SIS class, create a method to retrieve all courses assigned to a specific teacher.

main.py | SIS.py | Student.py | Course.py | Enrollment.py | Teacher.py | Payment.py

```python
from datetime import datetime
from Enrollment import Enrollment
from Student import Student
from Teacher import Teacher
from Payment import Payment
5 usages
class SIS:
    def __init__(self):
        self.students = []
        self.courses = []
        self.teachers = []
    3 usages
    def enroll_student_in_course(self, student, course):
        student.enroll_in_course(course)
        enrollment = Enrollment(student.student_id, course.course_id, datetime.now())
        course.enrollments.append(enrollment)
    3 usages
    def assign_teacher_to_course(self, teacher, course):
        course.assign_teacher(teacher)
        teacher.assigned_courses.append(course)
    3 usages
    def record_payment(self, student, amount, payment_date):
        payment = Payment(student.student_id, amount, payment_date)
        student.payments.append(payment)
    2 usages
    def generate_enrollment_report(self, course):
        print(f"Enrollment Report for Course {course.course_name}")
        for enrollment in course.enrollments:
            print(f"Student ID: {enrollment.student_id}, Enrollment Date: {enrollment.enrollment_date}")
    2 usages
    def generate_payment_report(self, student):
        print(f"Payment Report for Student {student.first_name} {student.last_name}")
        for payment in student.payments:
            print(f"Payment ID: {payment.payment_id}, Amount: {payment.amount}, Payment Date: {payment.payment_date}")
```

Task 7: Database Connectivity

Database Initialization:

Implement a method that initializes a database connection and creates tables for storing student, course, enrollment, teacher, and payment information. Create SQL scripts or use code-first migration to create tables with appropriate schemas for your SIS.

```python
import mysql.connector
class DatabaseConnector:
    def __init__(self):
        self.connection = None
    def open_connection(self):
        self.connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="Sushant@9546",
            database="SIS5"
        )
        self.create_database()
        self.create_tables()
    def close_connection(self):
        if self.connection:
            self.connection.close()
```

Data Insertion and Updating:

Implement methods to insert new data (e.g., enrollments, payments) into the database and update existing data (e.g., student information). Use methods to perform data insertion and updating. Implement validation checks to ensure data integrity and handle any errors during these operations.

```python
        self.connection.close()

    def create_tables(self):
        def create_tables(self):
            student_table_query = """
            CREATE TABLE IF NOT EXISTS students (
                id INT AUTO_INCREMENT PRIMARY KEY,
                first_name VARCHAR(255) NOT NULL,
                last_name VARCHAR(255) NOT NULL,
                birth_date DATE NOT NULL,
                email VARCHAR(255) NOT NULL,
                phone VARCHAR(20) NOT NULL
            );
            """
            course_table_query = """
            CREATE TABLE IF NOT EXISTS courses (
                id INT AUTO_INCREMENT PRIMARY KEY,
                course_name VARCHAR(255) NOT NULL,
                course_code VARCHAR(20) NOT NULL,
                instructor VARCHAR(255) NOT NULL
            );
```

main.py | DatabaseConnector.py | student.py | course.py | enrollment.py | teacher.py | payment.py

```python
        """
        teacher_table_query = """
        CREATE TABLE IF NOT EXISTS teachers (
            id INT AUTO_INCREMENT PRIMARY KEY,
            first_name VARCHAR(255) NOT NULL,
            last_name VARCHAR(255) NOT NULL,
            email VARCHAR(255) NOT NULL,
            assigned_courses Varchar(255) Not null
        );
        """
        payment_table_query = """
        CREATE TABLE IF NOT EXISTS payments (
            id INT AUTO_INCREMENT PRIMARY KEY,
            student_id INT NOT NULL,
            amount FLOAT NOT NULL,
            payment_date DATE NOT NULL,
            FOREIGN KEY (student_id) REFERENCES students(id)
        );
        """
        enrollment_table_query = """
        CREATE TABLE IF NOT EXISTS enrollments (
            id INT AUTO_INCREMENT PRIMARY KEY,
            student_id INT NOT NULL,
            course_id INT NOT NULL,
            enrollment_date DATE NOT NULL,
            FOREIGN KEY (student_id) REFERENCES students(id),
            FOREIGN KEY (course_id) REFERENCES courses(id)
        );
```

main.py | DatabaseConnector.py | student.py | course.py | enrollment.py | teacher.py | payment.py

```python
    def insert_enrollment(self):
        student_id = input("Enter student ID for enrollment: ")

        check_student_query = "SELECT id FROM students WHERE id = %s"
        self.cursor.execute(check_student_query, (student_id,))
        result = self.cursor.fetchone()

        if result is None:
            print(f"Error: Student with ID {student_id} does not exist.")
            return

        course_id = input("Enter course ID for enrollment: ")

        check_course_query = "SELECT id FROM courses WHERE id = %s"
        self.cursor.execute(check_course_query, (course_id,))
        result = self.cursor.fetchone()

        if result is None:
            print(f"Error: Course with ID {course_id} does not exist.")
            return

        enrollment_date = get_date_input()

        query = """
        INSERT INTO enrollments (student_id, course_id, enrollment_date)
```

main.py | DatabaseConnector.py | student.py | course.py | enrollment.py | teacher.py | payment.py

```python
        enrollment_date = get_date_input()

        query = """
        INSERT INTO enrollments (student_id, course_id, enrollment_date)
        VALUES (%s, %s, %s);
        """
        values = (student_id, course_id, enrollment_date)
        self.execute_query(query, values)
        print("Enrollment successful.")

    1 usage (1 dynamic)
    def assign_teacher_to_course(self):
        course_id = input("Enter course ID: ")
        teacher_id = input("Enter teacher ID: ")

        check_course_query = "SELECT id FROM courses WHERE id = %s"
        self.cursor.execute(check_course_query, (course_id,))
        result_course = self.cursor.fetchone()

        check_teacher_query = "SELECT id FROM teachers WHERE id = %s"
        self.cursor.execute(check_teacher_query, (teacher_id,))
        result_teacher = self.cursor.fetchone()

        if result_course is None:
            print(f"Error: Course with ID {course_id} does not exist.")
            return
        if result_teacher is None:
            print(f"Error: Teacher with ID {teacher_id} does not exist.")
```

main.py | DatabaseConnector.py | student.py | course.py | enrollment.py | teacher.py | payment.py

```python
            return
        update_course_query = "UPDATE courses SET instructor = %s WHERE id = %s"
        values = (teacher_id, course_id)
        self.execute_query(update_course_query, values)
        print("Teacher assigned to the course.")

    1 usage (1 dynamic)
    def insert_payment(self):
        student_id = input("Enter student ID for payment: ")
        amount = float(input("Enter payment amount: "))
        payment_date = get_date_input()
        check_student_query = "SELECT id FROM students WHERE id = %s"
        self.cursor.execute(check_student_query, (student_id,))
        result = self.cursor.fetchone()

        if result is None:
            print(f"Error: Student with ID {student_id} does not exist.")
            return
        query = """
        INSERT INTO payments (student_id, amount, payment_date)
        VALUES (%s, %s, %s);
        """
        values = (student_id, amount, payment_date)
        self.execute_query(query, values)
        print("Payment recorded successfully.")

    1 usage (1 dynamic)
    def get_student_info(self, student_id):
```

```python
    def get_student_info(self, student_id):

        check_student_query = "SELECT * FROM students WHERE id = %s"
        self.cursor.execute(check_student_query, (student_id,))
        result = self.cursor.fetchone()

        if result is None:
            print(f"Error: Student with ID {student_id} does not exist.")
            return None

        student_info = Student(result[0], result[1], result[2], result[3], result[4], result[5])
        return student_info
    1 usage (1 dynamic)
    def get_payment_history(self, student_id):

        query = """
            SELECT payments.id, payments.amount, payments.payment_date
            FROM payments
            WHERE payments.student_id = %s;
        """
        self.cursor.execute(query, (student_id,))
        payment_history_data = self.cursor.fetchall()

        payment_history = []
        for payment_data in payment_history_data:
            payment = Payment(*payment_data)
            payment_history.append(payment)
```

main.py | DatabaseConnector.py | student.py | course.py | enrollment.py | teacher.py | payment.py

```python
        return payment_history

    1 usage (1 dynamic)
    def get_enrolled_courses(self, student_id):

        query = """
            SELECT courses.course_id, courses.course_name, courses.course_code, courses.instructor_name
            FROM enrollments
            JOIN courses ON enrollments.course_id = courses.course_id
            WHERE enrollments.student_id = %s;
        """

        self.cursor.execute(query, (student_id,))
        enrolled_courses_data = self.cursor.fetchall()

        enrolled_courses = []
        for course_data in enrolled_courses_data:
            course = Course(*course_data)
            enrolled_courses.append(course)

        return enrolled_courses
    1 usage (1 dynamic)
    def insert_student(self, first_name, last_name, birth_date, email, phone):
        query = """
        INSERT INTO students (first_name, last_name, birth_date, email, phone)
        VALUES (%s, %s, %s, %s, %s);
        """
```

```python
        return enrolled_courses
    1 usage (1 dynamic)
    def insert_student(self, first_name, last_name, birth_date, email, phone):
        query = """
        INSERT INTO students (first_name, last_name, birth_date, email, phone)
        VALUES (%s, %s, %s, %s, %s);
        """
        values = (first_name, last_name, birth_date, email, phone)
        self.execute_query(query, values)
        print("Student added successfully.")
    1 usage (1 dynamic)
    def insert_teacher(self, first_name, last_name, email, assigned_courses):
        try:

            query = "INSERT INTO teachers (first_name, last_name, email, assigned_courses) VALUES (%s, %s, %s, %s)"
            values = (first_name, last_name, email, assigned_courses)
            self.cursor.execute(query, values)
            self.connection.commit()
        except Exception as e:
            print(f"Error inserting teacher: {e}")
```

Task 8: Student Enrollment

In this task, a new student, John Doe, is enrolling in the SIS. The system needs to record John's information, including his personal details, and enroll him in a few courses. Database connectivity is required to store this information.

John Doe's details:

- First Name: John
- Last Name: Doe
- Date of Birth: 1995-08-15
- Email: john.doe@example.com
- Phone Number: 123-456-7890

```
C:\Users\ssush\PycharmProjects\SIS5\venv\Scripts\python.exe C:\Users\ssush\PycharmProjects\SIS5\main.py

1. Add a New Student
2. Enroll Student in a Course
3. Assign Teacher to a Course
4. Make Payment
5. Display Student Information
6. Display Enrolled Courses
7. Add New Teacher
8. Display Payment History
9. Exit
Enter your choice (1-10): 1
Enter student's first name: Jhon
Enter student's last name: Doe
Enter of birth date (YYYY-MM-DD): 1995-08-15
Enter student's email: jhon.doe@example.com
Enter student's phone number: 123-456-7890
Student added successfully.
Student added successfully.
```

Expected Data saved in Database Successfully

```
ysql> select * from students;
+----+------------+-----------+------------+---------------------+--------------+
| id | first_name | last_name | birth_date | email               | phone        |
+----+------------+-----------+------------+---------------------+--------------+
  1 | RAMA       | KUMAR     | 0500-01-22 | rama@gmail.com      | 9652458358   |
  2 | aman       | singh     | 2024-09-09 | aman@gmail.com      | 9652321459   |
  3 | Ramu       | raj       | 2024-01-30 | ramuraj@yahoo.com   | 965243585    |
  4 | Jhon       | Doe       | 1995-08-15 | jhon.doe@example.com | 123-456-7890 |
+----+------------+-----------+------------+---------------------+--------------+
rows in set (0.01 sec)
```

John is enrolling in the following courses:

• Course 2: Mathematics 101

The system should perform the following tasks:

• Create a new student record in the database.

• Enroll John in the specified courses by creating enrollment records in the database.

```
C:\Users\ssush\PycharmProjects\SIS5\venv\Scripts\python.exe C:\Users\ssush\PycharmProjects\SIS5\main.py

1. Add a New Student
2. Enroll Student in a Course
3. Assign Teacher to a Course
4. Make Payment
5. Display Student Information
6. Display Enrolled Courses
7. Add New Teacher
8. Display Payment History
9. Exit
Enter your choice (1-10): 2
Enter student ID for enrollment: 4
Enter course ID for enrollment: 1
Enter date (YYYY-MM-DD): 2024-02-02
Enrollment successful.
```

Expected Data saved in Database Successfully

```
ysql> select * from ENROLLMENTS;
----+------------+-----------+----------------+
 id | student_id | course_id | enrollment_date |
----+------------+-----------+----------------+
  2 |          1 |         1 | 2024-01-30     |
  3 |          1 |         1 | 2024-01-30     |
  4 |          4 |         1 | 2024-02-02     |
----+------------+-----------+----------------+
 rows in set (0.01 sec)

ysql> select * from COURSES;
----+-------------+-------------+---------------+
 id | course_name | course_code | instructor    |
----+-------------+-------------+---------------+
  1 | Mathematics | MATH101     | Dr. RAMANUJAN |
----+-------------+-------------+---------------+
 row in set (0.01 sec)
```

Task 9: Teacher Assignment

In this task, a new teacher, Sarah Smith, is assigned to teach a course. The system needs to update the course record to reflect the teacher assignment.

Teacher's Details:

- Name: Sarah Smith
- Email: sarah.smith@example.com
- Expertise: Computer Science

The system should perform the following tasks:

- Retrieve the course record from the database based on the course code.
- Assign Sarah Smith as the instructor for the course.
- Update the course record in the database with the new instructor information.

```
1. Add a New Student
2. Enroll Student in a Course
3. Assign Teacher to a Course
4. Make Payment
5. Display Student Information
6. Display Enrolled Courses
7. Add New Teacher
8. Display Payment History
9. Exit
Enter your choice (1-10): 7
Enter teacher's first name: SARHA
Enter teacher's last name: SMITH
Enter teacher's email: sarh.smith@example.com
Enter teacher's assigned_courses: Computer Science
New teacher added successfully.
```

Expected Data saved in Database Successfully

```
ysql> select * from teachers;
----+------------+-----------+-------------------------+------------------+
 id | first_name | last_name | email                   | assigned_courses |
----+------------+-----------+-------------------------+------------------+
  1 | DR         | RAMANUJAN | dr.raama@yahoo.com      | Mathematics      |
  2 | SARAH      | SMITHS    | sarah.smiths@example.com | Computer Science |
  3 | anu        | singh     | anu@gmail.com           | english          |
----+------------+-----------+-------------------------+------------------+
3 rows in set (0.00 sec)

ysql>
```

Task 10: Payment Record

In this task, a student, Jane Johnson, makes a payment for her enrolled courses. The system needs to record this payment in the database.

Jane Johnson's details:

- Student ID: 101
- Payment Amount: $500.00
- Payment Date: 2023-04-10

The system should perform the following tasks:

- Retrieve Jane Johnson's student record from the database based on her student ID.
- Record the payment information in the database, associating it with Jane's student record.
- Update Jane's outstanding balance in the database based on the payment amount.

```
Run    main
C:\Users\ssush\PycharmProjects\SIS5\venv\Scripts\python.exe C:\Users\ssush\PycharmProjects\SIS5\main.py

1. Add a New Student
2. Enroll Student in a Course
3. Assign Teacher to a Course
4. Make Payment
5. Display Student Information
6. Display Enrolled Courses
7. Add New Teacher
8. Display Payment History
9. Exit
Enter your choice (1-10): 4
Enter student ID for payment: 3
Enter payment amount: 500.00
Enter date (YYYY-MM-DD): 2023-04-10
Payment recorded successfully.
```

Expected Data saved in Database Successfully

```
mysql> select * from payments;
+----+------------+--------+--------------+
| id | student_id | amount | payment_date |
+----+------------+--------+--------------+
|  1 |          1 |   5000 | 2024-02-03   |
|  2 |          3 |  95240 | 2024-03-10   |
|  3 |          1 |   5000 | 2024-02-03   |
|  4 |          3 |    500 | 2023-04-10   |
+----+------------+--------+--------------+
4 rows in set (0.00 sec)

mysql>
```

**********Thankyou**********