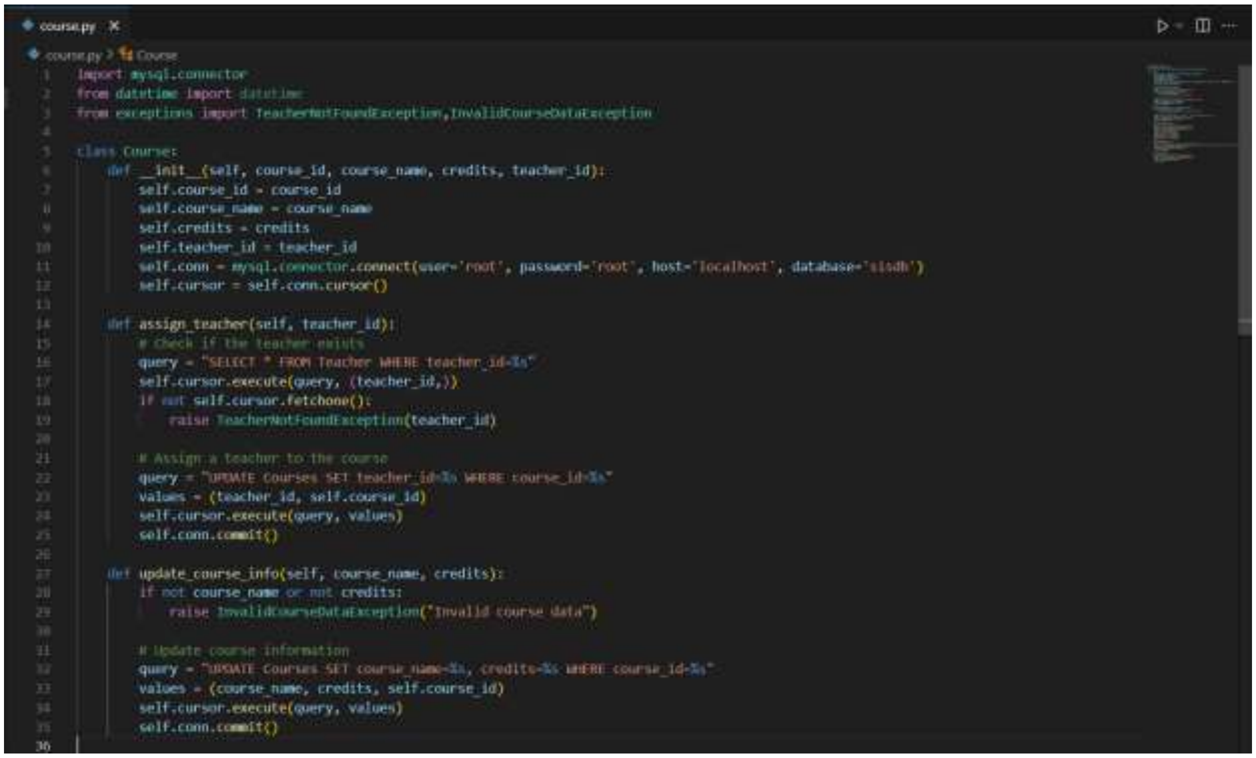
Assignment 2

Student Information System

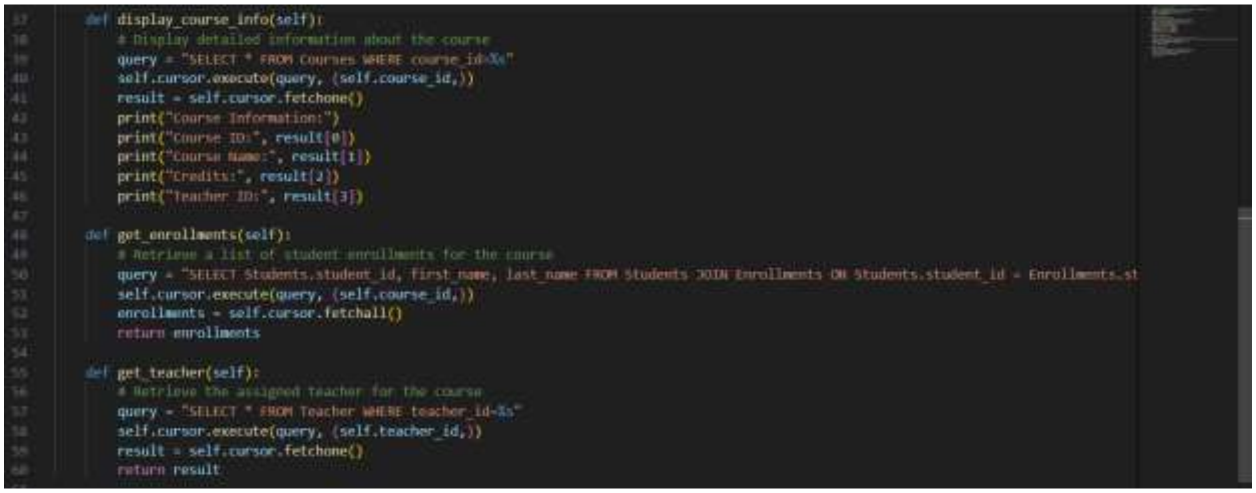
Task(1,2,3):



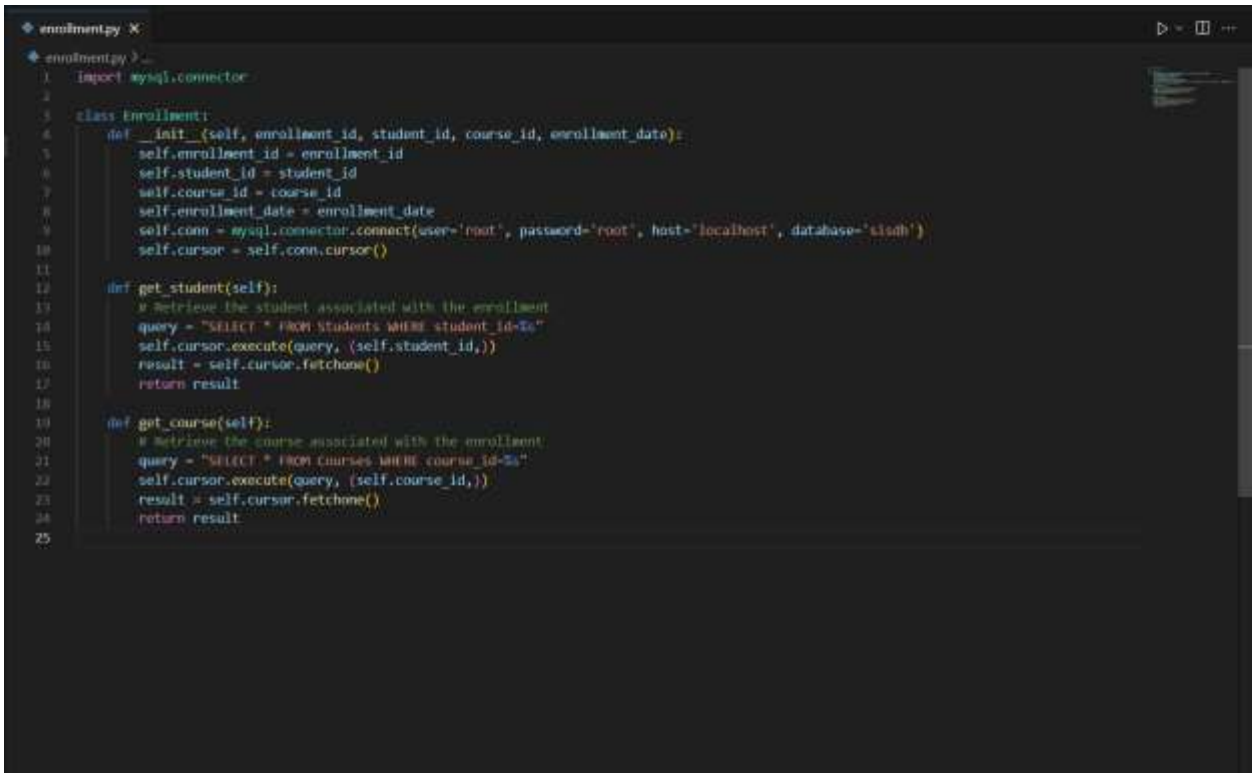
course.py

course.py > Course

```python
import mysql.connector
from datetime import datetime
from exceptions import TeacherNotFoundException,InvalidCourseDataException

class Course:
    def __init__(self, course_id, course_name, credits, teacher_id):
        self.course_id = course_id
        self.course_name = course_name
        self.credits = credits
        self.teacher_id = teacher_id
        self.conn = mysql.connector.connect(user='root', password='root', host='localhost', database='sisdb')
        self.cursor = self.conn.cursor()

    def assign_teacher(self, teacher_id):
        # Check if the teacher exists
        query = "SELECT * FROM Teacher WHERE teacher_id=%s"
        self.cursor.execute(query, (teacher_id,))
        if not self.cursor.fetchone():
            raise TeacherNotFoundException(teacher_id)

        # Assign a teacher to the course
        query = "UPDATE Courses SET teacher_id=%s WHERE course_id=%s"
        values = (teacher_id, self.course_id)
        self.cursor.execute(query, values)
        self.conn.commit()

    def update_course_info(self, course_name, credits):
        if not course_name or not credits:
            raise InvalidCourseDataException("Invalid course data")

        # Update course information
        query = "UPDATE Courses SET course_name=%s, credits=%s WHERE course_id=%s"
        values = (course_name, credits, self.course_id)
        self.cursor.execute(query, values)
        self.conn.commit()

```



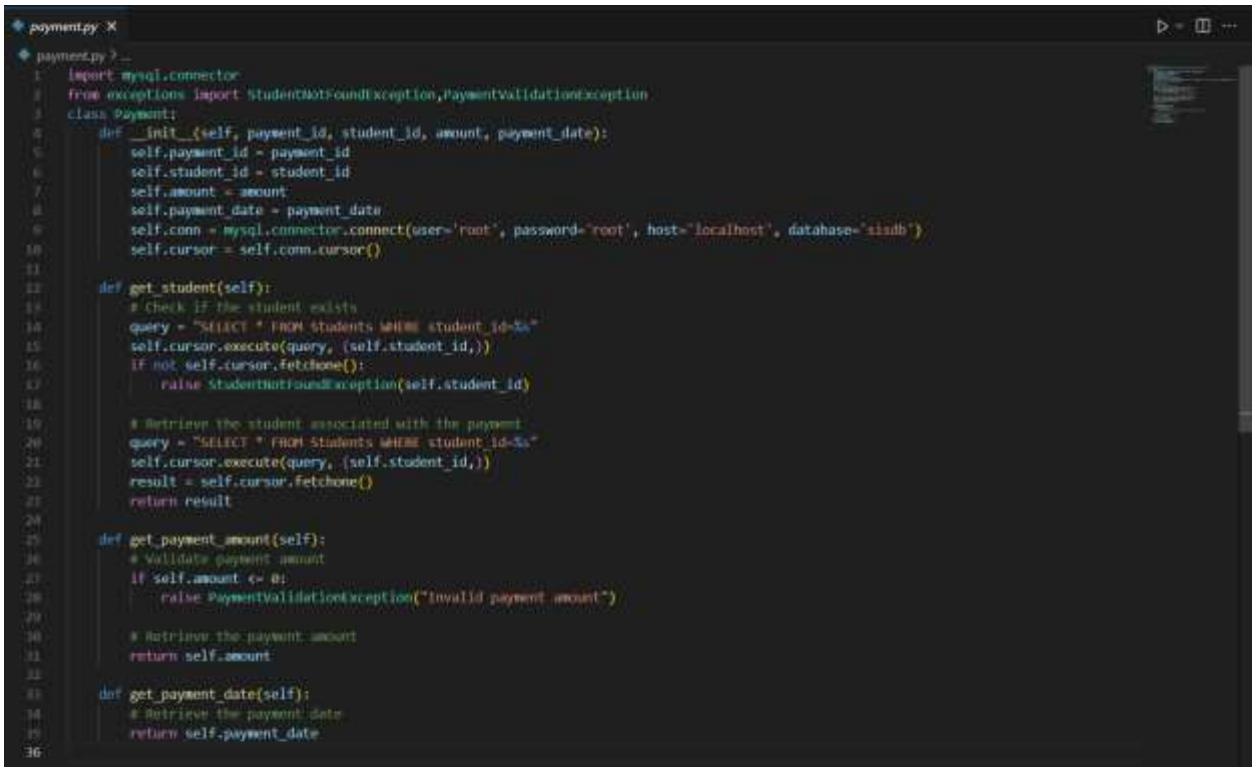
```python
    def display_course_info(self):
        # Display detailed information about the course
        query = "SELECT * FROM Courses WHERE course_id=%s"
        self.cursor.execute(query, (self.course_id,))
        result = self.cursor.fetchone()
        print("Course Information:")
        print("Course ID:", result[0])
        print("Course Name:", result[1])
        print("Credits:", result[2])
        print("Teacher ID:", result[3])

    def get_enrollments(self):
        # Retrieve a list of student enrollments for the course
        query = "SELECT Students.student_id, first_name, last_name FROM Students JOIN Enrollments ON Students.student_id = Enrollments.st
        self.cursor.execute(query, (self.course_id,))
        enrollments = self.cursor.fetchall()
        return enrollments

    def get_teacher(self):
        # Retrieve the assigned teacher for the course
        query = "SELECT * FROM Teacher WHERE teacher_id=%s"
        self.cursor.execute(query, (self.teacher_id,))
        result = self.cursor.fetchone()
        return result
```



enrollment.py

enrollment.py > ...

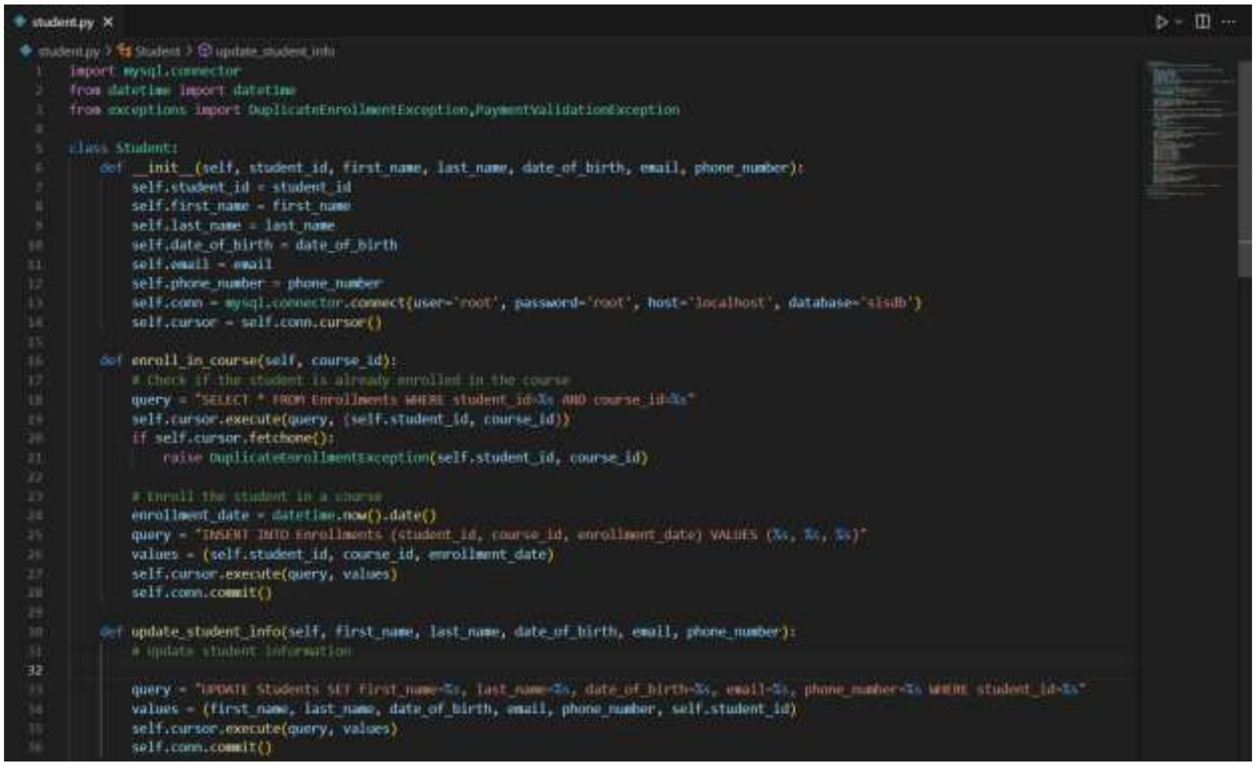
```python
import mysql.connector

class Enrollment:
    def __init__(self, enrollment_id, student_id, course_id, enrollment_date):
        self.enrollment_id = enrollment_id
        self.student_id = student_id
        self.course_id = course_id
        self.enrollment_date = enrollment_date
        self.conn = mysql.connector.connect(user='root', password='root', host='localhost', database='sisdb')
        self.cursor = self.conn.cursor()

    def get_student(self):
        # Retrieve the student associated with the enrollment
        query = "SELECT * FROM Students WHERE student_id=%s"
        self.cursor.execute(query, (self.student_id,))
        result = self.cursor.fetchone()
        return result

    def get_course(self):
        # Retrieve the course associated with the enrollment
        query = "SELECT * FROM Courses WHERE course_id=%s"
        self.cursor.execute(query, (self.course_id,))
        result = self.cursor.fetchone()
        return result

```



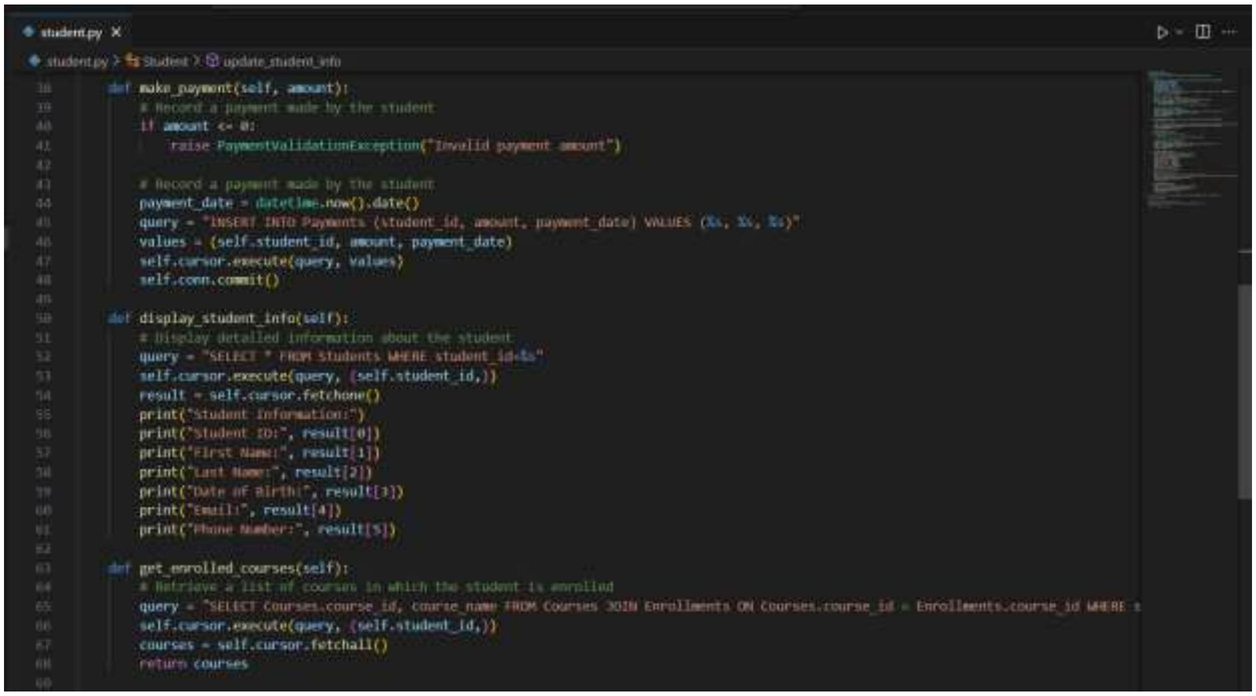
payment.py

payment.py > ...

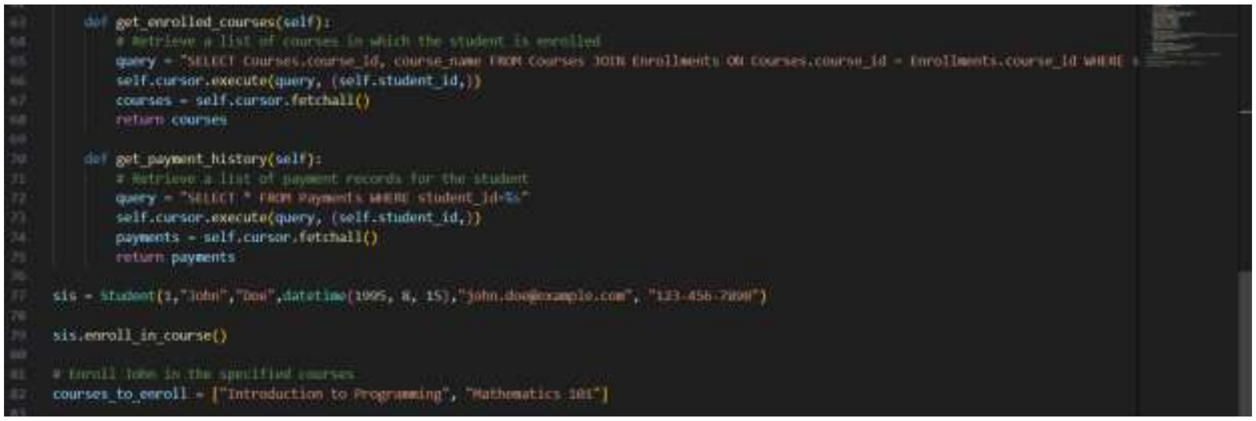
```python
import mysql.connector
from exceptions import StudentNotFoundException,PaymentValidationException
class Payment:
    def __init__(self, payment_id, student_id, amount, payment_date):
        self.payment_id = payment_id
        self.student_id = student_id
        self.amount = amount
        self.payment_date = payment_date
        self.conn = mysql.connector.connect(user='root', password='root', host='localhost', database='sisdb')
        self.cursor = self.conn.cursor()

    def get_student(self):
        # Check if the student exists
        query = "SELECT * FROM Students WHERE student_id=%s"
        self.cursor.execute(query, (self.student_id,))
        if not self.cursor.fetchone():
            raise StudentNotFoundException(self.student_id)

        # Retrieve the student associated with the payment
        query = "SELECT * FROM Students WHERE student_id=%s"
        self.cursor.execute(query, (self.student_id,))
        result = self.cursor.fetchone()
        return result

    def get_payment_amount(self):
        # Validate payment amount
        if self.amount <= 0:
            raise PaymentValidationException("Invalid payment amount")

        # Retrieve the payment amount
        return self.amount

    def get_payment_date(self):
        # Retrieve the payment date
        return self.payment_date

```



```python
import mysql.connector
from datetime import datetime
from exceptions import DuplicateEnrollmentException,PaymentValidationException

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.conn = mysql.connector.connect(user='root', password='root', host='localhost', database='sisdb')
        self.cursor = self.conn.cursor()

    def enroll_in_course(self, course_id):
        # Check if the student is already enrolled in the course
        query = "SELECT * FROM Enrollments WHERE student_id=%s AND course_id=%s"
        self.cursor.execute(query, (self.student_id, course_id))
        if self.cursor.fetchone():
            raise DuplicateEnrollmentException(self.student_id, course_id)

        # Enroll the student in a course
        enrollment_date = datetime.now().date()
        query = "INSERT INTO Enrollments (student_id, course_id, enrollment_date) VALUES (%s, %s, %s)"
        values = (self.student_id, course_id, enrollment_date)
        self.cursor.execute(query, values)
        self.conn.commit()

    def update_student_info(self, first_name, last_name, date_of_birth, email, phone_number):
        # update student information

        query = "UPDATE Students SET first_name=%s, last_name=%s, date_of_birth=%s, email=%s, phone_number=%s WHERE student_id=%s"
        values = (first_name, last_name, date_of_birth, email, phone_number, self.student_id)
        self.cursor.execute(query, values)
        self.conn.commit()
```

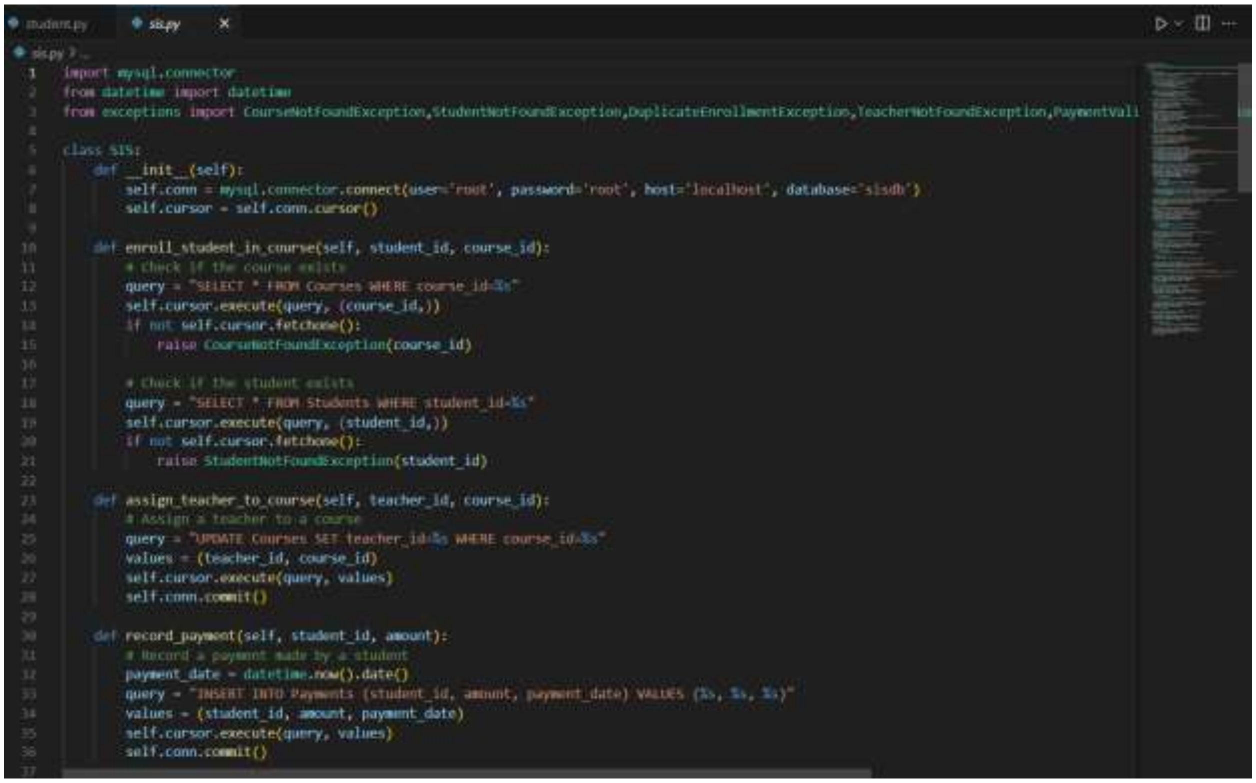


```python
    def make_payment(self, amount):
        # Record a payment made by the student
        if amount <= 0:
            raise PaymentValidationException("Invalid payment amount")

        # Record a payment made by the student
        payment_date = datetime.now().date()
        query = "INSERT INTO Payments (student_id, amount, payment_date) VALUES (%s, %s, %s)"
        values = (self.student_id, amount, payment_date)
        self.cursor.execute(query, values)
        self.conn.commit()

    def display_student_info(self):
        # Display detailed information about the student
        query = "SELECT * FROM Students WHERE student_id=%s"
        self.cursor.execute(query, (self.student_id,))
        result = self.cursor.fetchone()
        print("Student Information:")
        print("Student ID:", result[0])
        print("First Name:", result[1])
        print("Last Name:", result[2])
        print("Date of Birth:", result[3])
        print("Email:", result[4])
        print("Phone Number:", result[5])

    def get_enrolled_courses(self):
        # Retrieve a list of courses in which the student is enrolled
        query = "SELECT Courses.course_id, course_name FROM Courses JOIN Enrollments ON Courses.course_id = Enrollments.course_id WHERE s
        self.cursor.execute(query, (self.student_id,))
        courses = self.cursor.fetchall()
        return courses

```

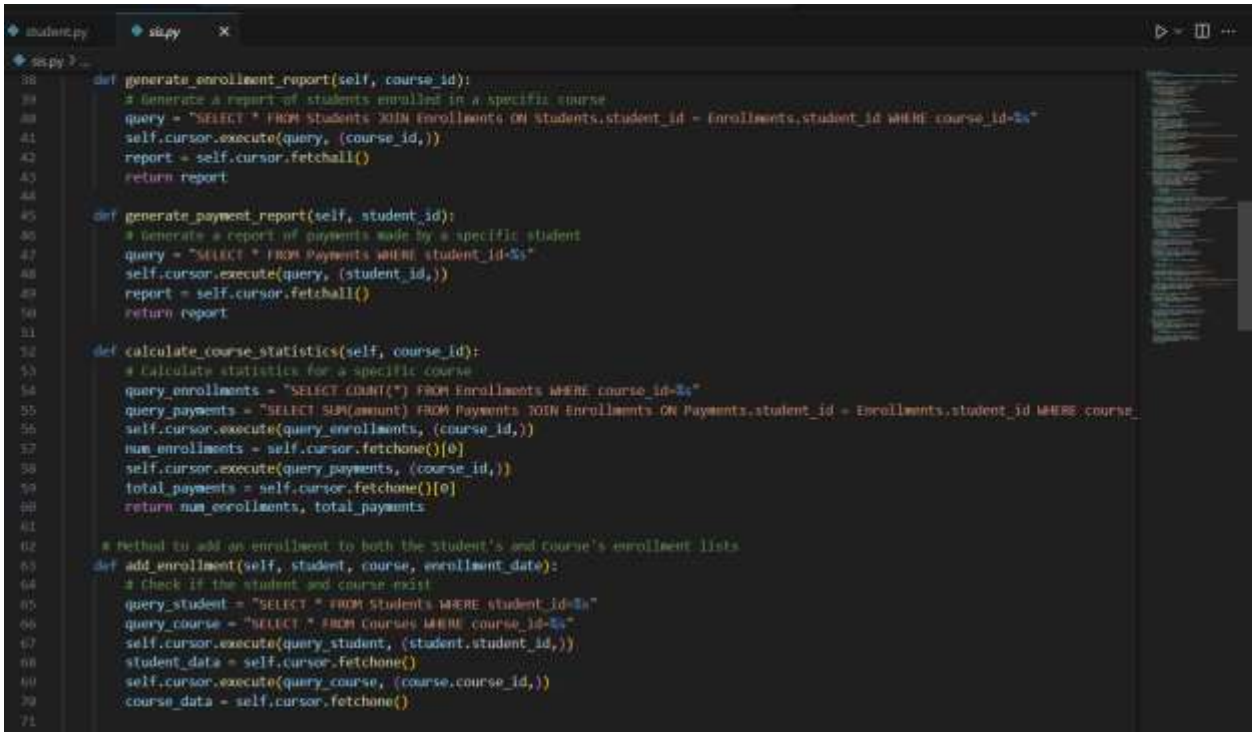


```python
    def get_enrolled_courses(self):
        # Retrieve a list of courses in which the student is enrolled
        query = "SELECT Courses.course_id, course_name FROM Courses JOIN Enrollments ON Courses.course_id = Enrollments.course_id WHERE s
        self.cursor.execute(query, (self.student_id,))
        courses = self.cursor.fetchall()
        return courses

    def get_payment_history(self):
        # Retrieve a list of payment records for the student
        query = "SELECT * FROM Payments WHERE student_id=%s"
        self.cursor.execute(query, (self.student_id,))
        payments = self.cursor.fetchall()
        return payments

sis = Student(1,"John","Doe",datetime(1995, 8, 15),"john.doe@example.com", "123-456-7890")

sis.enroll_in_course()

# Enroll John in the specified courses
courses_to_enroll = ["Introduction to Programming", "Mathematics 101"]
```

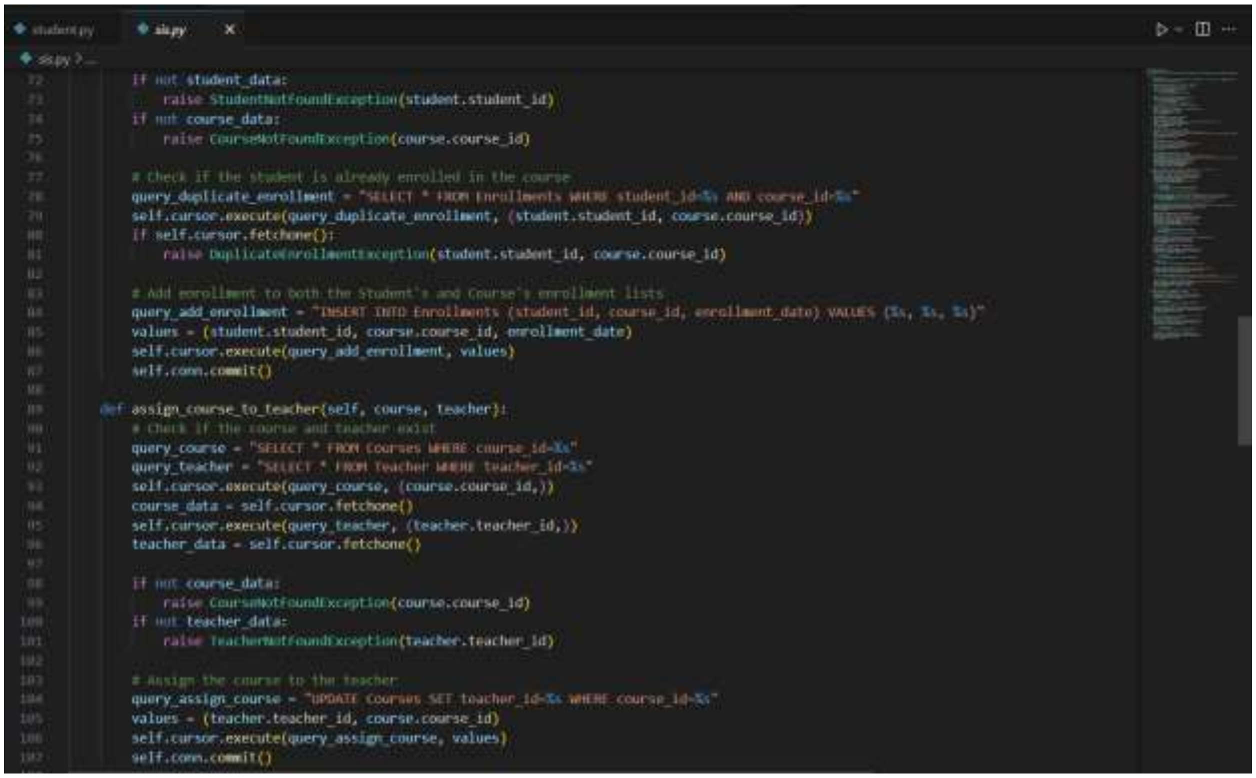
SIS:



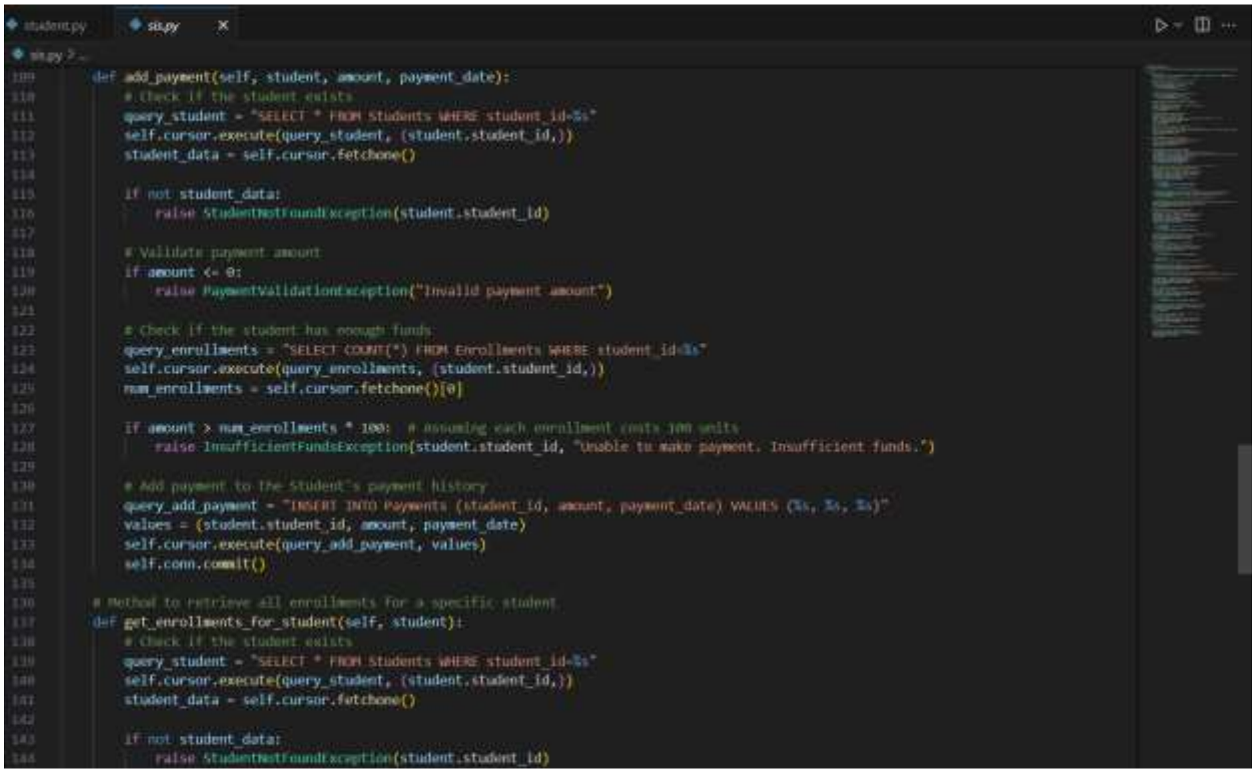
```python
import mysql.connector
from datetime import datetime
from exceptions import CourseNotFoundException,StudentNotFoundException,DuplicateEnrollmentException,TeacherNotFoundException,PaymentVali

class SIS:
    def __init__(self):
        self.conn = mysql.connector.connect(user='root', password='root', host='localhost', database='sisdb')
        self.cursor = self.conn.cursor()

    def enroll_student_in_course(self, student_id, course_id):
        # Check if the course exists
        query = "SELECT * FROM Courses WHERE course_id=%s"
        self.cursor.execute(query, (course_id,))
        if not self.cursor.fetchone():
            raise CourseNotFoundException(course_id)

        # Check if the student exists
        query = "SELECT * FROM Students WHERE student_id=%s"
        self.cursor.execute(query, (student_id,))
        if not self.cursor.fetchone():
            raise StudentNotFoundException(student_id)

    def assign_teacher_to_course(self, teacher_id, course_id):
        # Assign a teacher to a course
        query = "UPDATE Courses SET teacher_id=%s WHERE course_id=%s"
        values = (teacher_id, course_id)
        self.cursor.execute(query, values)
        self.conn.commit()

    def record_payment(self, student_id, amount):
        # Record a payment made by a student
        payment_date = datetime.now().date()
        query = "INSERT INTO Payments (student_id, amount, payment_date) VALUES (%s, %s, %s)"
        values = (student_id, amount, payment_date)
        self.cursor.execute(query, values)
        self.conn.commit()
```



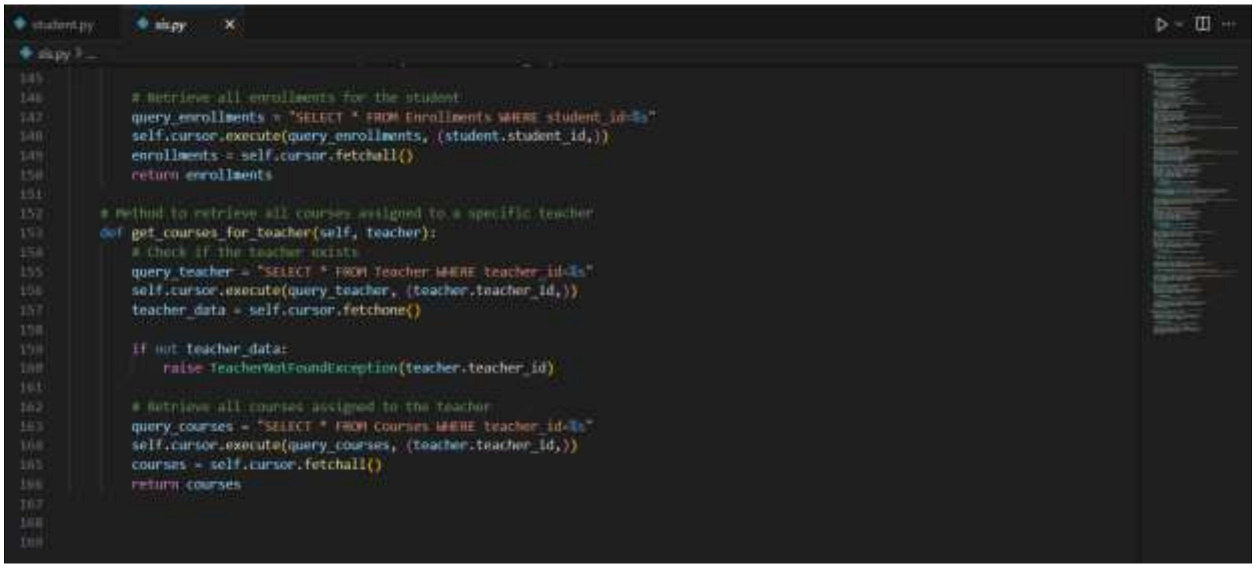
```
    def generate_enrollment_report(self, course_id):
        # Generate a report of students enrolled in a specific course
        query = "SELECT * FROM Students JOIN Enrollments ON Students.student_id = Enrollments.student_id WHERE course_id=%s"
        self.cursor.execute(query, (course_id,))
        report = self.cursor.fetchall()
        return report

    def generate_payment_report(self, student_id):
        # Generate a report of payments made by a specific student
        query = "SELECT * FROM Payments WHERE student_id=%s"
        self.cursor.execute(query, (student_id,))
        report = self.cursor.fetchall()
        return report

    def calculate_course_statistics(self, course_id):
        # Calculate statistics for a specific course
        query_enrollments = "SELECT COUNT(*) FROM Enrollments WHERE course_id=%s"
        query_payments = "SELECT SUM(amount) FROM Payments JOIN Enrollments ON Payments.student_id = Enrollments.student_id WHERE course_
        self.cursor.execute(query_enrollments, (course_id,))
        num_enrollments = self.cursor.fetchone()[0]
        self.cursor.execute(query_payments, (course_id,))
        total_payments = self.cursor.fetchone()[0]
        return num_enrollments, total_payments

     # Method to add an enrollment to both the Student's and Course's enrollment lists
    def add_enrollment(self, student, course, enrollment_date):
        # Check if the student and course exist
        query_student = "SELECT * FROM Students WHERE student_id=%s"
        query_course = "SELECT * FROM Courses WHERE course_id=%s"
        self.cursor.execute(query_student, (student.student_id,))
        student_data = self.cursor.fetchone()
        self.cursor.execute(query_course, (course.course_id,))
        course_data = self.cursor.fetchone()

```



```
        if not student_data:
            raise StudentNotFoundException(student.student_id)
        if not course_data:
            raise CourseNotFoundException(course.course_id)

        # Check if the student is already enrolled in the course
        query_duplicate_enrollment = "SELECT * FROM Enrollments WHERE student_id=%s AND course_id=%s"
        self.cursor.execute(query_duplicate_enrollment, (student.student_id, course.course_id))
        if self.cursor.fetchone():
            raise DuplicateEnrollmentException(student.student_id, course.course_id)

        # Add enrollment to both the Student's and Course's enrollment lists
        query_add_enrollment = "INSERT INTO Enrollments (student_id, course_id, enrollment_date) VALUES (%s, %s, %s)"
        values = (student.student_id, course.course_id, enrollment_date)
        self.cursor.execute(query_add_enrollment, values)
        self.conn.commit()

    def assign_course_to_teacher(self, course, teacher):
        # Check if the course and teacher exist
        query_course = "SELECT * FROM Courses WHERE course_id=%s"
        query_teacher = "SELECT * FROM Teacher WHERE teacher_id=%s"
        self.cursor.execute(query_course, (course.course_id,))
        course_data = self.cursor.fetchone()
        self.cursor.execute(query_teacher, (teacher.teacher_id,))
        teacher_data = self.cursor.fetchone()

        if not course_data:
            raise CourseNotFoundException(course.course_id)
        if not teacher_data:
            raise TeacherNotFoundException(teacher.teacher_id)

        # Assign the course to the teacher
        query_assign_course = "UPDATE Courses SET teacher_id=%s WHERE course_id=%s"
        values = (teacher.teacher_id, course.course_id)
        self.cursor.execute(query_assign_course, values)
        self.conn.commit()
```

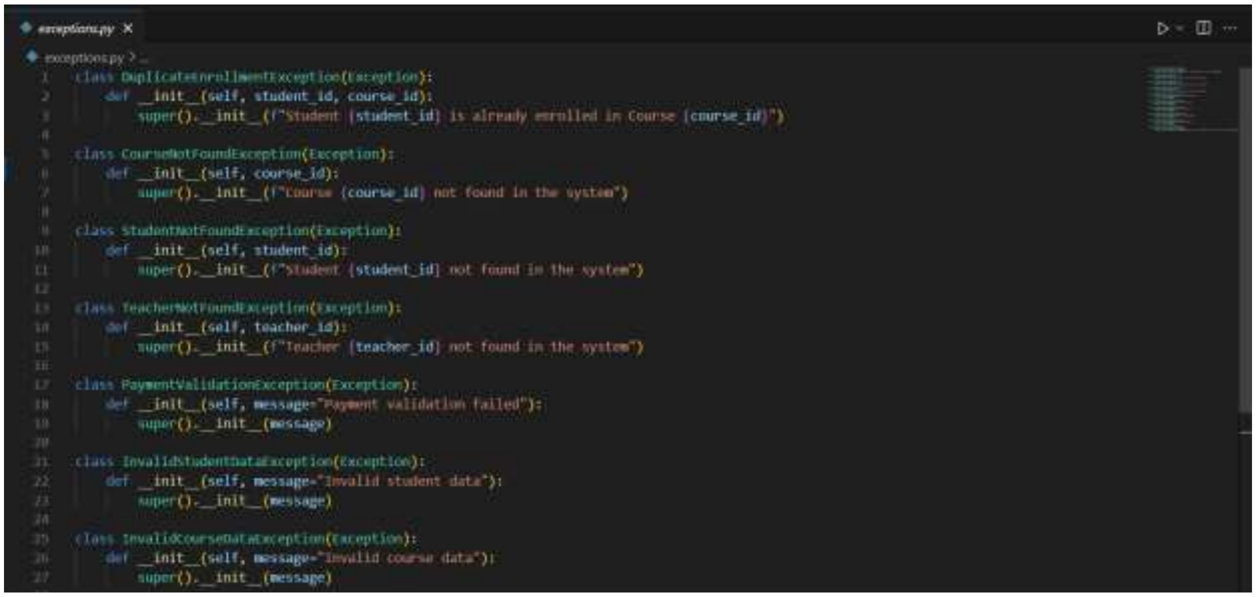


```python
    def add_payment(self, student, amount, payment_date):
        # Check if the student exists
        query_student = "SELECT * FROM Students WHERE student_id=%s"
        self.cursor.execute(query_student, (student.student_id,))
        student_data = self.cursor.fetchone()

        if not student_data:
            raise StudentNotFoundException(student.student_id)

        # Validate payment amount
        if amount <= 0:
            raise PaymentValidationException("Invalid payment amount")

        # Check if the student has enough funds
        query_enrollments = "SELECT COUNT(*) FROM Enrollments WHERE student_id=%s"
        self.cursor.execute(query_enrollments, (student.student_id,))
        num_enrollments = self.cursor.fetchone()[0]

        if amount > num_enrollments * 100:  # assuming each enrollment costs 100 units
            raise InsufficientFundsException(student.student_id, "Unable to make payment. Insufficient funds.")

        # Add payment to the Student's payment history
        query_add_payment = "INSERT INTO Payments (student_id, amount, payment_date) VALUES (%s, %s, %s)"
        values = (student.student_id, amount, payment_date)
        self.cursor.execute(query_add_payment, values)
        self.conn.commit()

    # Method to retrieve all enrollments for a specific student
    def get_enrollments_for_student(self, student):
        # Check if the student exists
        query_student = "SELECT * FROM Students WHERE student_id=%s"
        self.cursor.execute(query_student, (student.student_id,))
        student_data = self.cursor.fetchone()

        if not student_data:
            raise StudentNotFoundException(student.student_id)
```



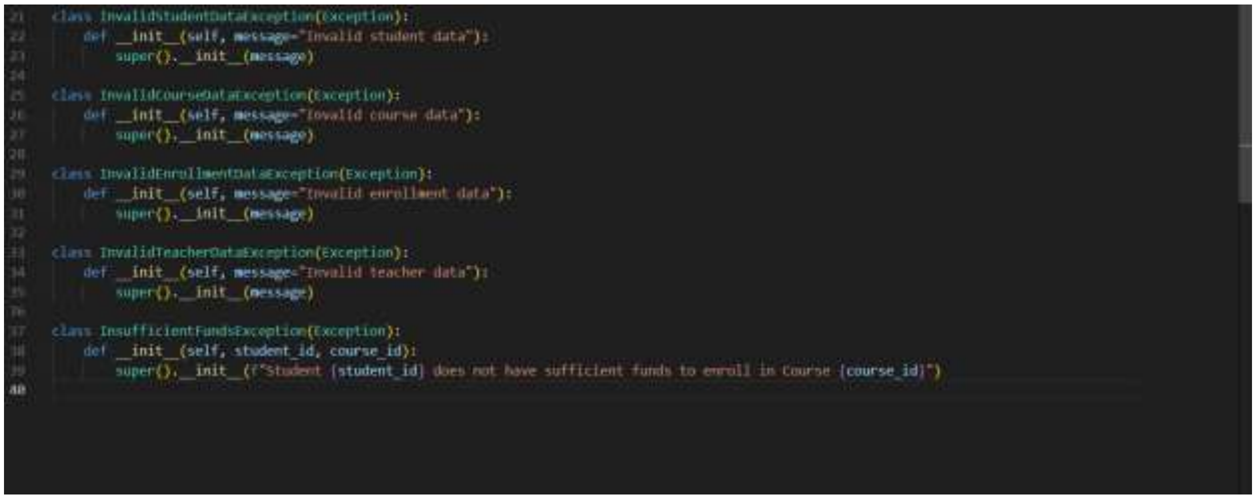
```python

        # Retrieve all enrollments for the student
        query_enrollments = "SELECT * FROM Enrollments WHERE student_id=%s"
        self.cursor.execute(query_enrollments, (student.student_id,))
        enrollments = self.cursor.fetchall()
        return enrollments

    # Method to retrieve all courses assigned to a specific teacher
    def get_courses_for_teacher(self, teacher):
        # Check if the teacher exists
        query_teacher = "SELECT * FROM Teacher WHERE teacher_id=%s"
        self.cursor.execute(query_teacher, (teacher.teacher_id,))
        teacher_data = self.cursor.fetchone()

        if not teacher_data:
            raise TeacherNotFoundException(teacher.teacher_id)

        # Retrieve all courses assigned to the teacher
        query_courses = "SELECT * FROM Courses WHERE teacher_id=%s"
        self.cursor.execute(query_courses, (teacher.teacher_id,))
        courses = self.cursor.fetchall()
        return courses
```

Taks(4,5,6)Exceptions:



exceptions.py

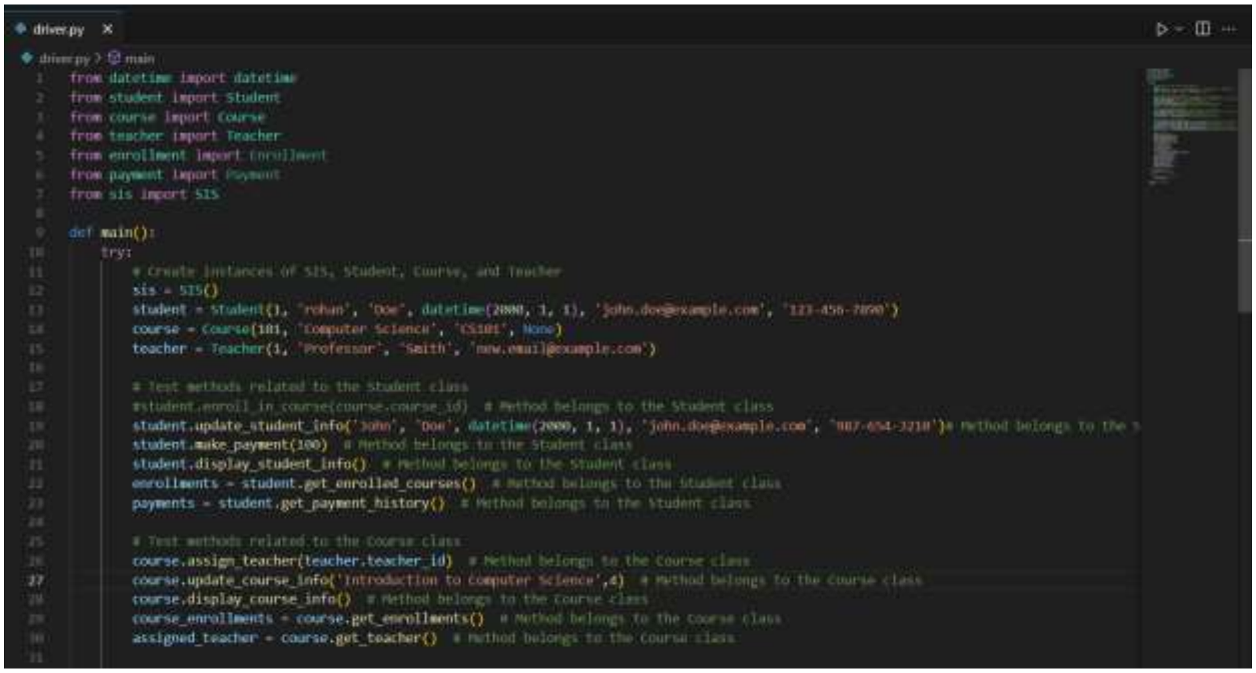
```python
class DuplicateEnrollmentException(Exception):
    def __init__(self, student_id, course_id):
        super().__init__(f"Student {student_id} is already enrolled in Course {course_id}")

class CourseNotFoundException(Exception):
    def __init__(self, course_id):
        super().__init__(f"Course {course_id} not found in the system")

class StudentNotFoundException(Exception):
    def __init__(self, student_id):
        super().__init__(f"Student {student_id} not found in the system")

class TeacherNotFoundException(Exception):
    def __init__(self, teacher_id):
        super().__init__(f"Teacher {teacher_id} not found in the system")

class PaymentValidationException(Exception):
    def __init__(self, message="Payment validation failed"):
        super().__init__(message)

class InvalidStudentDataException(Exception):
    def __init__(self, message="Invalid student data"):
        super().__init__(message)

class InvalidCourseDataException(Exception):
    def __init__(self, message="Invalid course data"):
        super().__init__(message)
```



```python
class InvalidStudentDataException(Exception):
    def __init__(self, message="Invalid student data"):
        super().__init__(message)

class InvalidCourseDataException(Exception):
    def __init__(self, message="Invalid course data"):
        super().__init__(message)
```

```python
class InvalidEnrollmentDataException(Exception):
    def __init__(self, message="Invalid enrollment data"):
        super().__init__(message)

class InvalidTeacherDataException(Exception):
    def __init__(self, message="Invalid teacher data"):
        super().__init__(message)

class InsufficientFundsException(Exception):
    def __init__(self, student_id, course_id):
        super().__init__(f"Student {student_id} does not have sufficient funds to enroll in Course {course_id}")
```

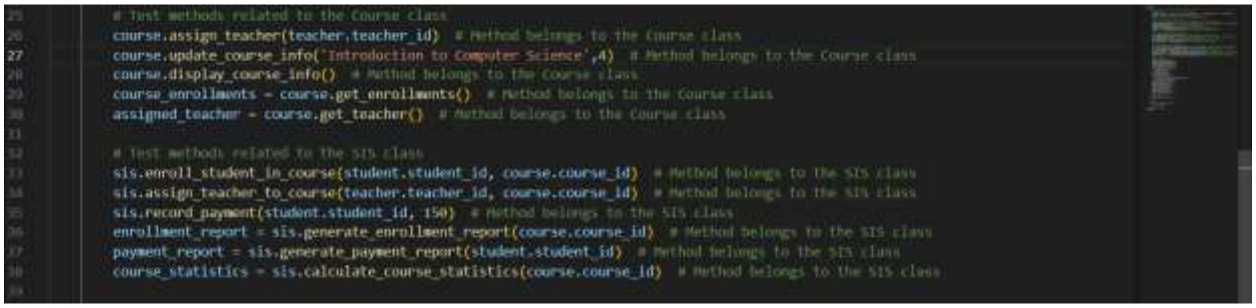
Driver Program:



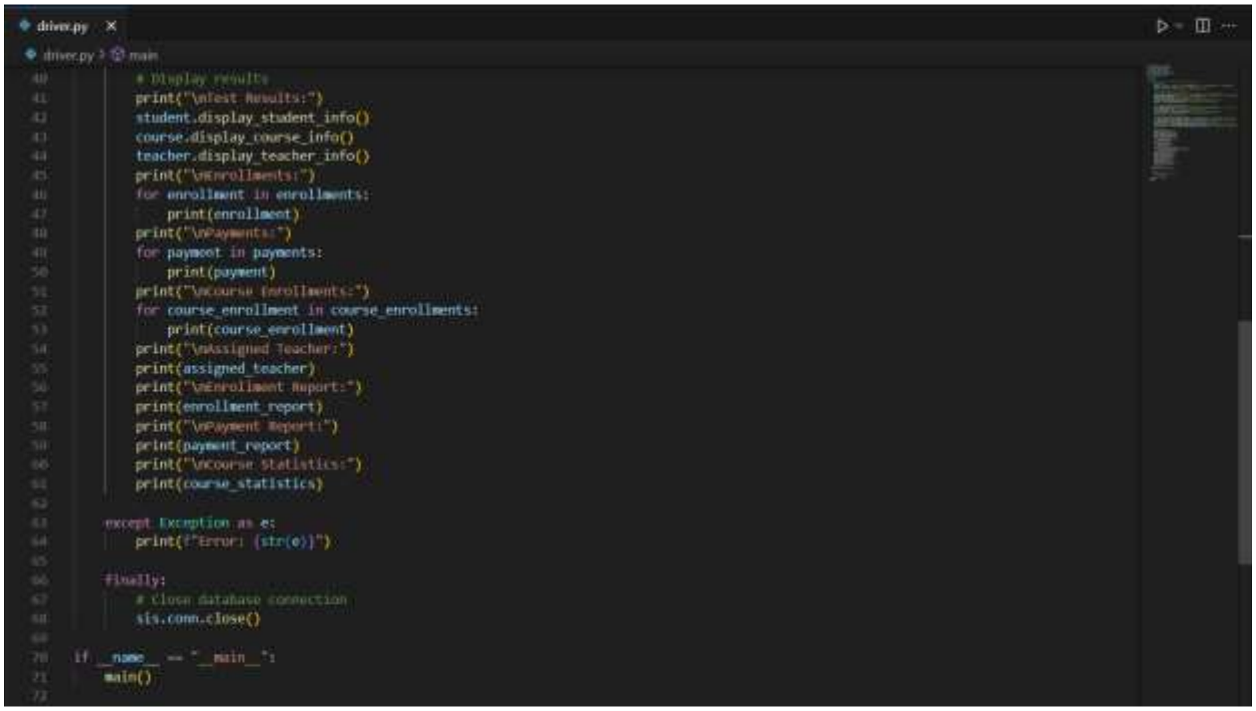
driver.py

driver.py > main

```python
from datetime import datetime
from student import Student
from course import Course
from teacher import Teacher
from enrollment import Enrollment
from payment import Payment
from sis import SIS

def main():
    try:
        # Create instances of SIS, Student, Course, and Teacher
        sis = SIS()
        student = Student(1, 'rohan', 'Doe', datetime(2000, 1, 1), 'john.doe@example.com', '123-456-7890')
        course = Course(101, 'Computer Science', 'CS101', None)
        teacher = Teacher(1, 'Professor', 'Smith', 'new.email@example.com')

        # Test methods related to the Student class
        #student.enroll_in_course(course.course_id)  # Method belongs to the Student class
        student.update_student_info('John', 'Doe', datetime(2000, 1, 1), 'john.doe@example.com', '987-654-3210')# Method belongs to the S
        student.make_payment(100)  # Method belongs to the Student class
        student.display_student_info()  # Method belongs to the Student class
        enrollments = student.get_enrolled_courses()  # Method belongs to the Student class
        payments = student.get_payment_history()  # Method belongs to the Student class

        # Test methods related to the Course class
        course.assign_teacher(teacher.teacher_id)  # Method belongs to the Course class
        course.update_course_info('Introduction to Computer Science',4)  # Method belongs to the Course class
        course.display_course_info()  # Method belongs to the Course class
        course_enrollments = course.get_enrollments()  # Method belongs to the Course class
        assigned_teacher = course.get_teacher()  # Method belongs to the Course class

```

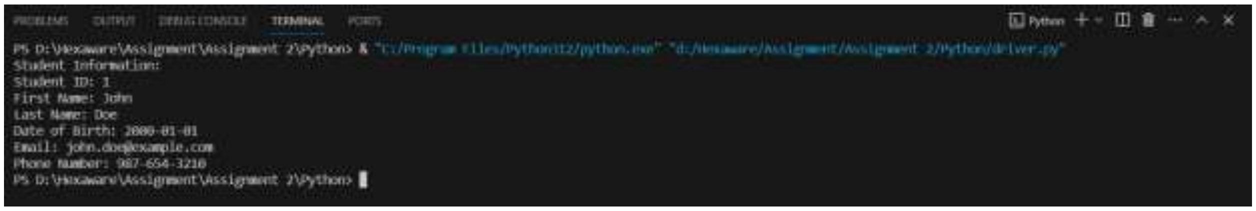


```python
        # Test methods related to the Course class
        course.assign_teacher(teacher.teacher_id)  # Method belongs to the Course class
        course.update_course_info('Introduction to Computer Science',4)  # Method belongs to the Course class
        course.display_course_info()  # Method belongs to the Course class
        course_enrollments = course.get_enrollments()  # Method belongs to the Course class
        assigned_teacher = course.get_teacher()  # Method belongs to the Course class

        # Test methods related to the SIS class
        sis.enroll_student_in_course(student.student_id, course.course_id)  # Method belongs to the SIS class
        sis.assign_teacher_to_course(teacher.teacher_id, course.course_id)  # Method belongs to the SIS class
        sis.record_payment(student.student_id, 150)  # Method belongs to the SIS class
        enrollment_report = sis.generate_enrollment_report(course.course_id)  # Method belongs to the SIS class
        payment_report = sis.generate_payment_report(student.student_id)  # Method belongs to the SIS class
        course_statistics = sis.calculate_course_statistics(course.course_id)  # Method belongs to the SIS class

```

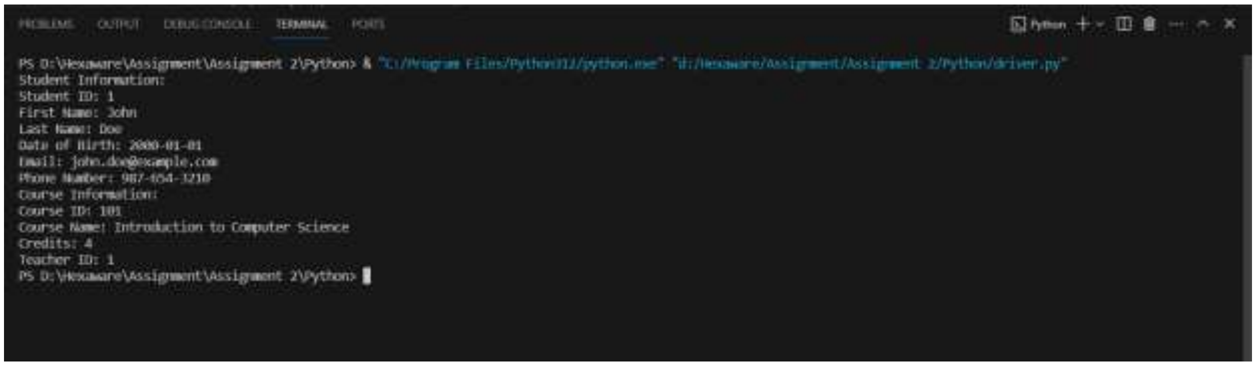


```python
            # Display results
            print("\nTest Results:")
            student.display_student_info()
            course.display_course_info()
            teacher.display_teacher_info()
            print("\nEnrollments:")
            for enrollment in enrollments:
                print(enrollment)
            print("\nPayments:")
            for payment in payments:
                print(payment)
            print("\nCourse Enrollments:")
            for course_enrollment in course_enrollments:
                print(course_enrollment)
            print("\nAssigned Teacher:")
            print(assigned_teacher)
            print("\nEnrollment Report:")
            print(enrollment_report)
            print("\nPayment Report:")
            print(payment_report)
            print("\nCourse Statistics:")
            print(course_statistics)

        except Exception as e:
            print(f"Error: {str(e)}")

        finally:
            # Close database connection
            sis.conn.close()

if __name__ == "__main__":
    main()
```

Outputs:

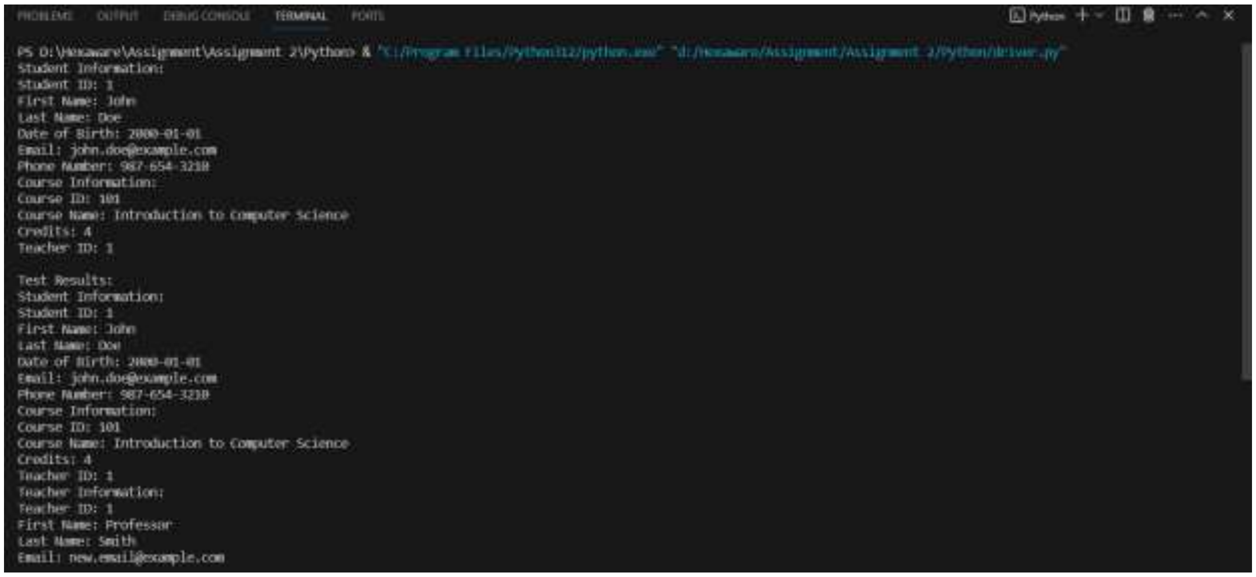


```
PS D:\Hexaware\Assignment\Assignment 2\Python> & "C:/Program Files/Python312/python.exe" "d:/Hexaware/Assignment/Assignment 2/Python/driver.py"
Student Information:
Student ID: 1
First Name: John
Last Name: Doe
Date of Birth: 2000-01-01
Email: john.doe@example.com
Phone Number: 987-654-3210
PS D:\Hexaware\Assignment\Assignment 2\Python>
```

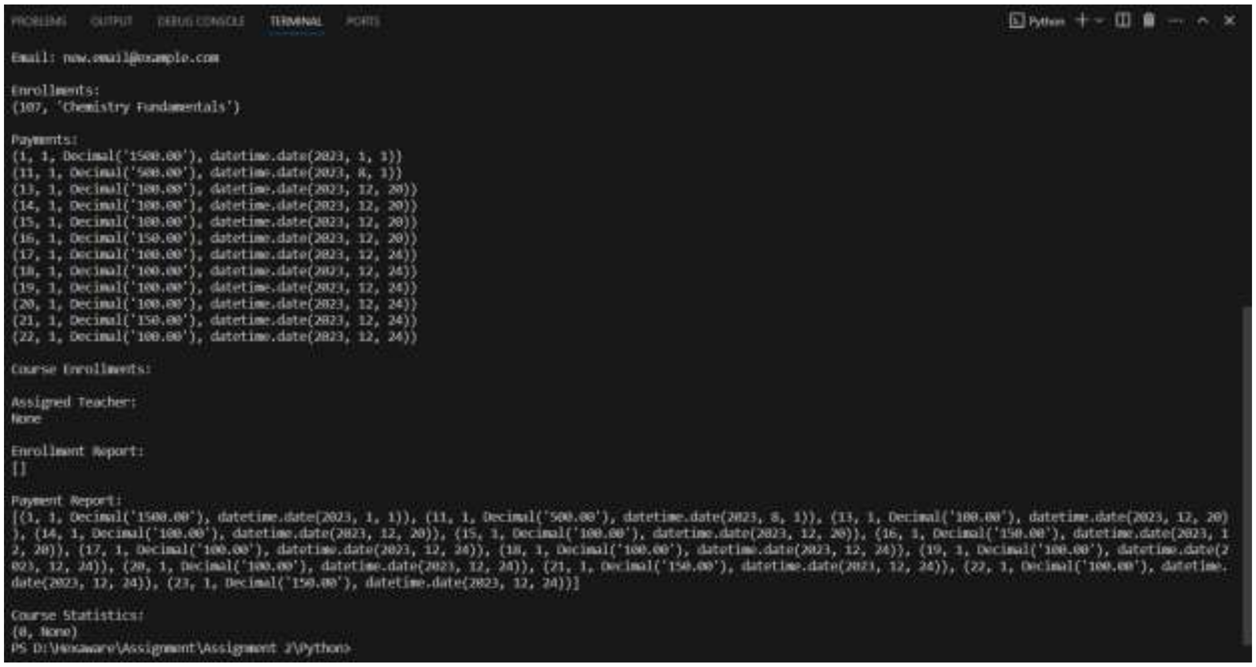


```
PS D:\Hexaware\Assignment\Assignment 2\Python> & "C:/Program Files/Python312/python.exe" "d:/Hexaware/Assignment/Assignment 2/Python/driver.py"
Student Information:
Student ID: 1
First Name: John
Last Name: Doe
Date of Birth: 2000-01-01
Email: john.doe@example.com
Phone Number: 987-654-3210
Course Information:
Course ID: 101
Course Name: Introduction to Computer Science
Credits: 4
Teacher ID: 1
PS D:\Hexaware\Assignment\Assignment 2\Python>
```

Complete Driver Program Outputs with collections:

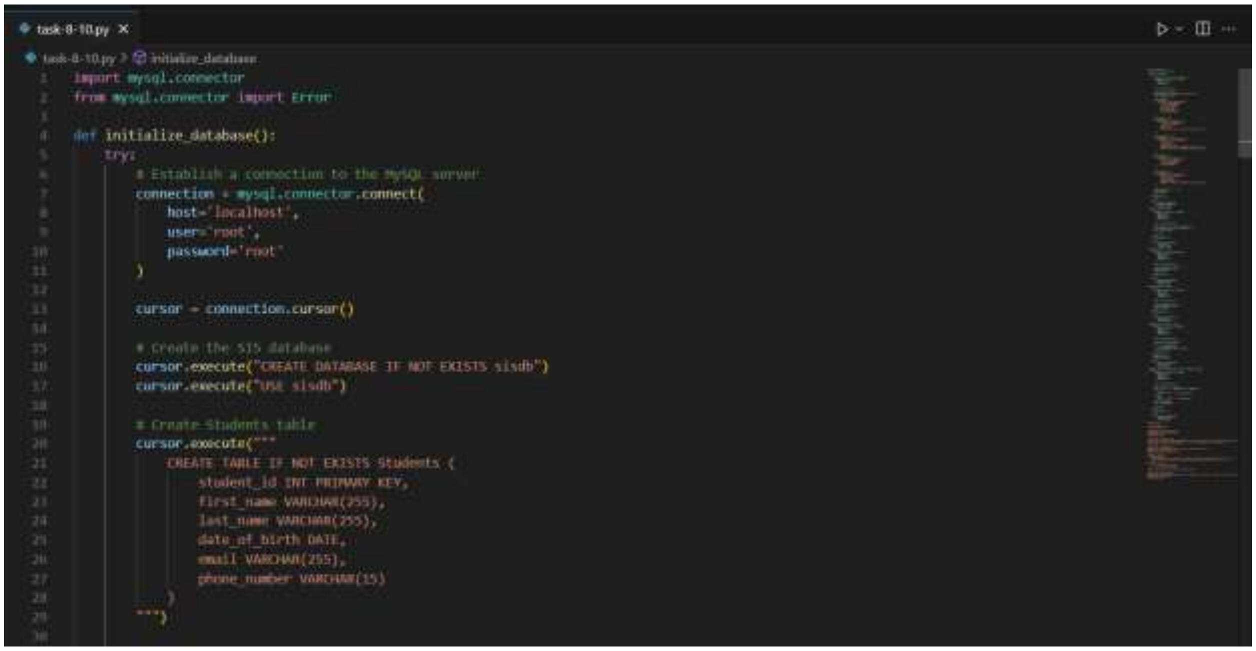


```
PS D:\Hexaware\Assignment\Assignment 2\Python> & "C:/Program Files/Python312/python.exe" "d:/Hexaware/Assignment/Assignment 2/Python/driver.py"
Student Information:
Student ID: 1
First Name: John
Last Name: Doe
Date of Birth: 2000-01-01
Email: john.doe@example.com
Phone Number: 987-654-3210
Course Information:
Course ID: 101
Course Name: Introduction to Computer Science
Credits: 4
Teacher ID: 1

Test Results:
Student Information:
Student ID: 1
First Name: John
Last Name: Doe
Date of Birth: 2000-01-01
Email: john.doe@example.com
Phone Number: 987-654-3210
Course Information:
Course ID: 101
Course Name: Introduction to Computer Science
Credits: 4
Teacher ID: 1
Teacher Information:
Teacher ID: 1
First Name: Professor
Last Name: Smith
Email: new.email@example.com
```



```
Email: new.email@example.com

Enrollments:
(107, 'Chemistry Fundamentals')

Payments:
(1, 1, Decimal('1500.00'), datetime.date(2023, 1, 1))
(11, 1, Decimal('500.00'), datetime.date(2023, 8, 1))
(13, 1, Decimal('100.00'), datetime.date(2023, 12, 20))
(14, 1, Decimal('100.00'), datetime.date(2023, 12, 20))
(15, 1, Decimal('100.00'), datetime.date(2023, 12, 20))
(16, 1, Decimal('150.00'), datetime.date(2023, 12, 20))
(17, 1, Decimal('100.00'), datetime.date(2023, 12, 24))
(18, 1, Decimal('100.00'), datetime.date(2023, 12, 24))
(19, 1, Decimal('100.00'), datetime.date(2023, 12, 24))
(20, 1, Decimal('100.00'), datetime.date(2023, 12, 24))
(21, 1, Decimal('150.00'), datetime.date(2023, 12, 24))
(22, 1, Decimal('100.00'), datetime.date(2023, 12, 24))

Course Enrollments:

Assigned Teacher:
None

Enrollment Report:
[]

Payment Report:
[(1, 1, Decimal('1500.00'), datetime.date(2023, 1, 1)), (11, 1, Decimal('500.00'), datetime.date(2023, 8, 1)), (13, 1, Decimal('100.00'), datetime.date(2023, 12, 20)
), (14, 1, Decimal('100.00'), datetime.date(2023, 12, 20)), (15, 1, Decimal('100.00'), datetime.date(2023, 12, 20)), (16, 1, Decimal('150.00'), datetime.date(2023, 1
2, 20)), (17, 1, Decimal('100.00'), datetime.date(2023, 12, 24)), (18, 1, Decimal('100.00'), datetime.date(2023, 12, 24)), (19, 1, Decimal('100.00'), datetime.date(2
023, 12, 24)), (20, 1, Decimal('100.00'), datetime.date(2023, 12, 24)), (21, 1, Decimal('150.00'), datetime.date(2023, 12, 24)), (22, 1, Decimal('100.00'), datetime.
date(2023, 12, 24)), (23, 1, Decimal('150.00'), datetime.date(2023, 12, 24))]

Course Statistics:
(0, None)
PS D:\Hexaware\Assignment\Assignment 2\Python>
```

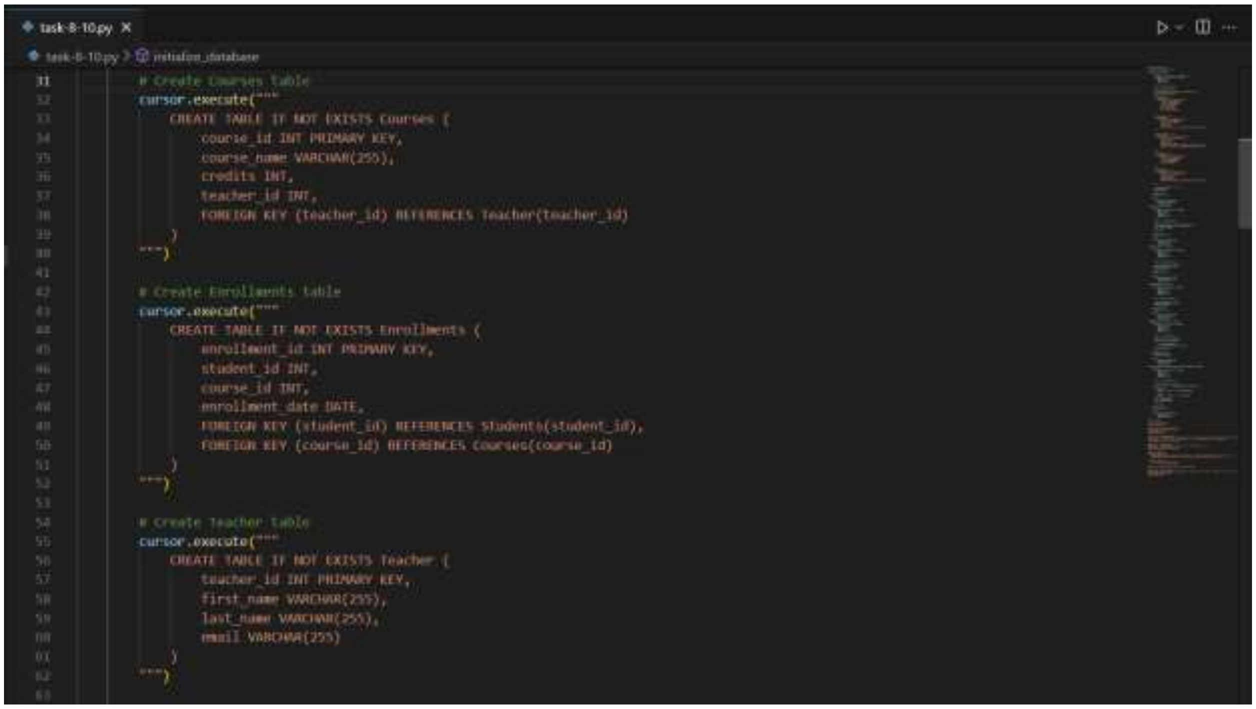
Task(7,8,9,10):



task-8-10.py ✕

task-8-10.py > initialize_database

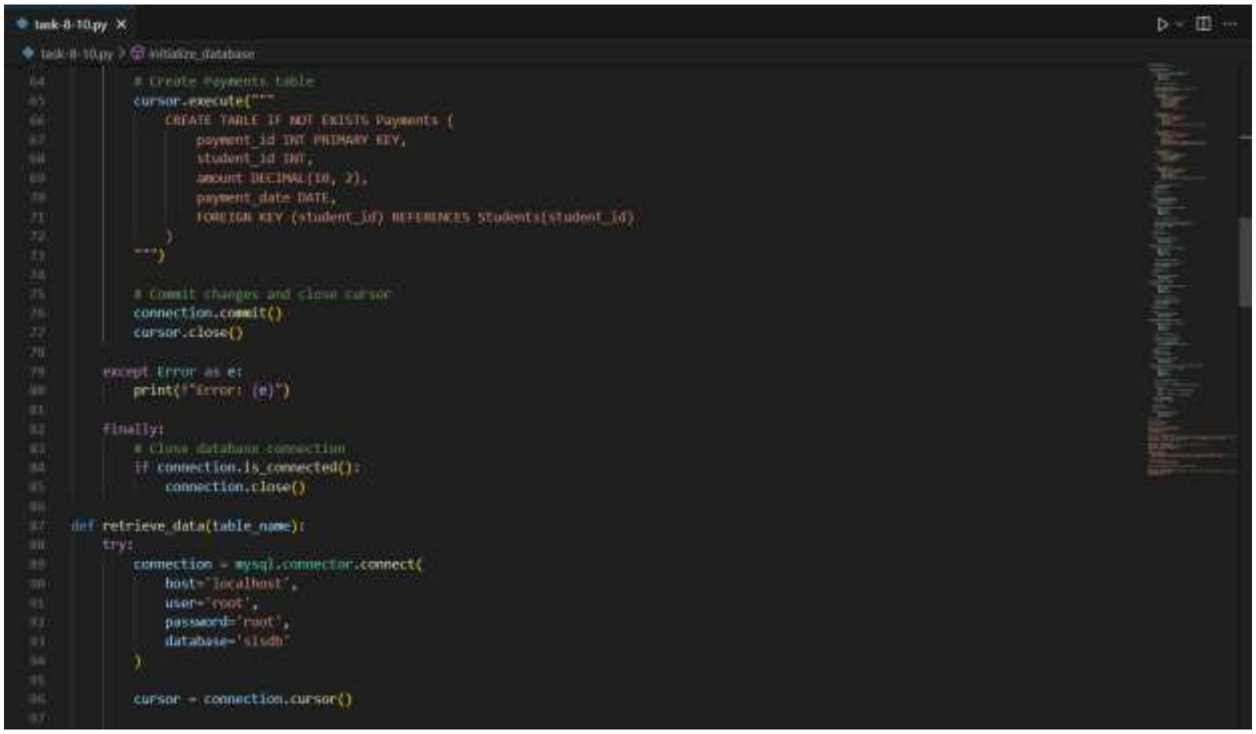
```python
import mysql.connector
from mysql.connector import Error

def initialize_database():
    try:
        # Establish a connection to the MySQL server
        connection = mysql.connector.connect(
            host='localhost',
            user='root',
            password='root'
        )

        cursor = connection.cursor()

        # Create the SIS database
        cursor.execute("CREATE DATABASE IF NOT EXISTS sisdb")
        cursor.execute("USE sisdb")

        # Create Students table
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS Students (
                student_id INT PRIMARY KEY,
                first_name VARCHAR(255),
                last_name VARCHAR(255),
                date_of_birth DATE,
                email VARCHAR(255),
                phone_number VARCHAR(15)
            )
        """)

```



task-8-10.py ✕

task-8-10.py > initialize_database

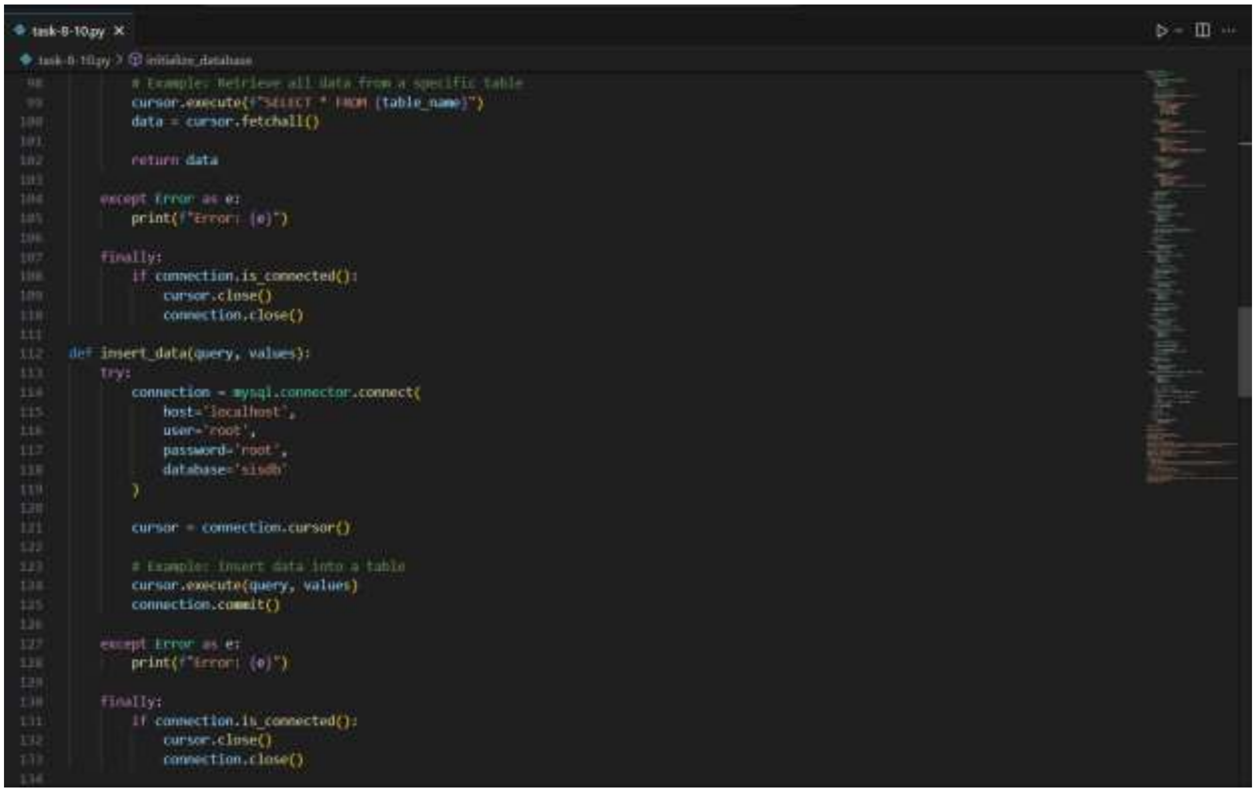
```python
        # Create Courses table
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS Courses (
                course_id INT PRIMARY KEY,
                course_name VARCHAR(255),
                credits INT,
                teacher_id INT,
                FOREIGN KEY (teacher_id) REFERENCES Teacher(teacher_id)
            )
        """)

        # Create Enrollments table
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS Enrollments (
                enrollment_id INT PRIMARY KEY,
                student_id INT,
                course_id INT,
                enrollment_date DATE,
                FOREIGN KEY (student_id) REFERENCES Students(student_id),
                FOREIGN KEY (course_id) REFERENCES Courses(course_id)
            )
        """)

        # Create Teacher table
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS Teacher (
                teacher_id INT PRIMARY KEY,
                first_name VARCHAR(255),
                last_name VARCHAR(255),
                email VARCHAR(255)
            )
        """)

```



task-8-10.py ×

task-8-10.py > initialize_database

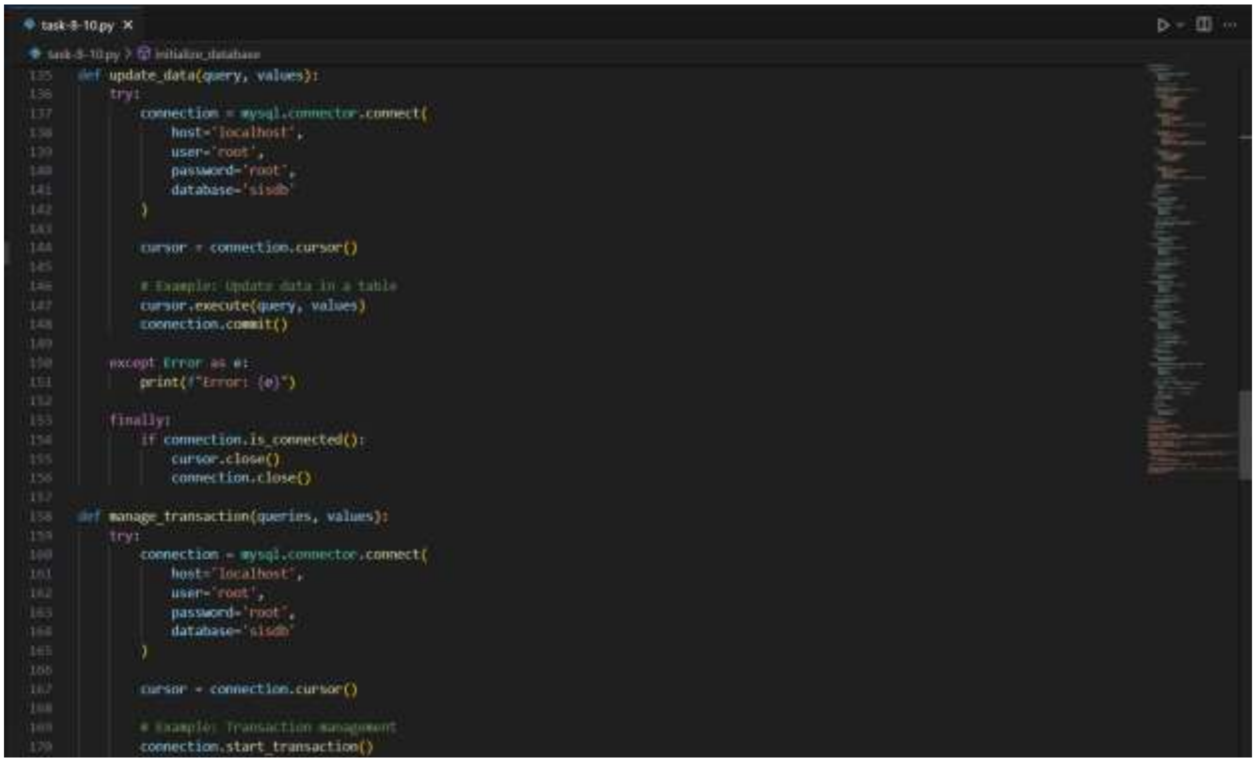
```python
        # Create Payments table
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS Payments (
                payment_id INT PRIMARY KEY,
                student_id INT,
                amount DECIMAL(10, 2),
                payment_date DATE,
                FOREIGN KEY (student_id) REFERENCES Students(student_id)
            )
        """)

        # Commit changes and close cursor
        connection.commit()
        cursor.close()

    except Error as e:
        print(f"Error: {e}")

    finally:
        # Close database connection
        if connection.is_connected():
            connection.close()

def retrieve_data(table_name):
    try:
        connection = mysql.connector.connect(
            host='localhost',
            user='root',
            password='root',
            database='sisdb'
        )

        cursor = connection.cursor()

```



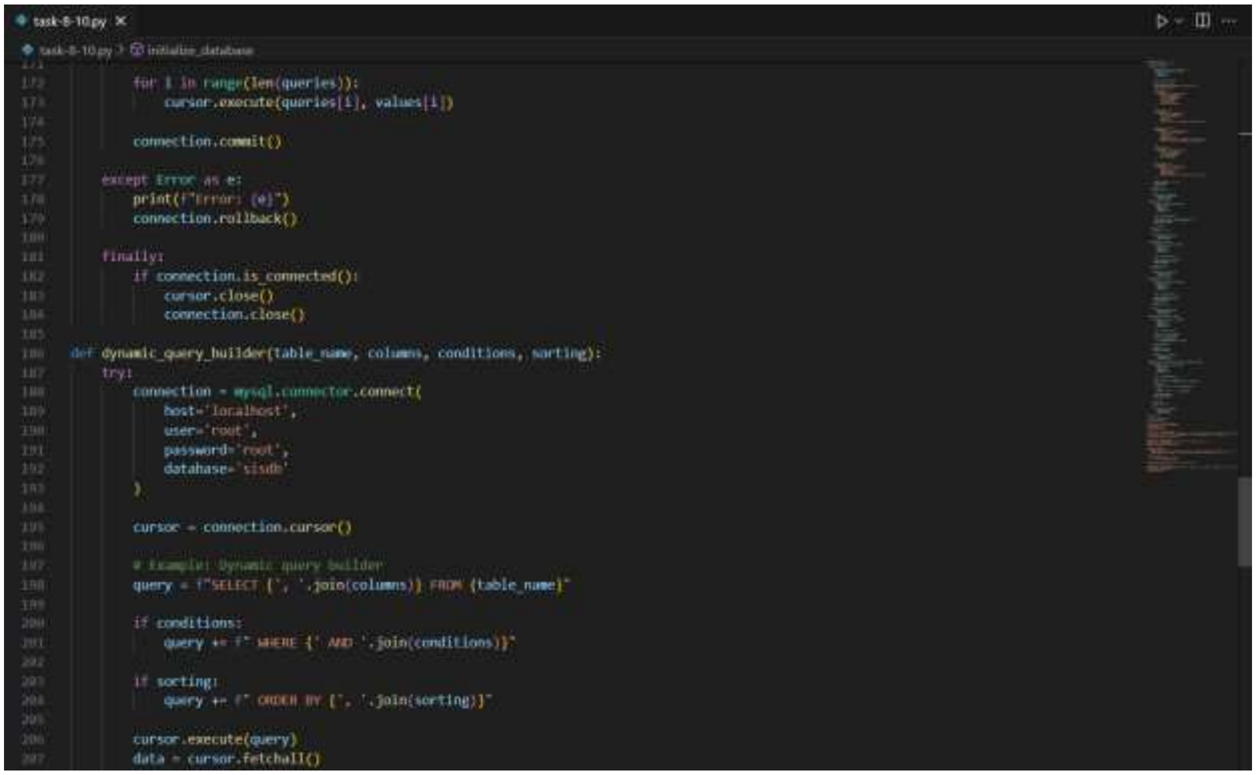
task-8-10.py ×

task-8-10.py > initialize_database

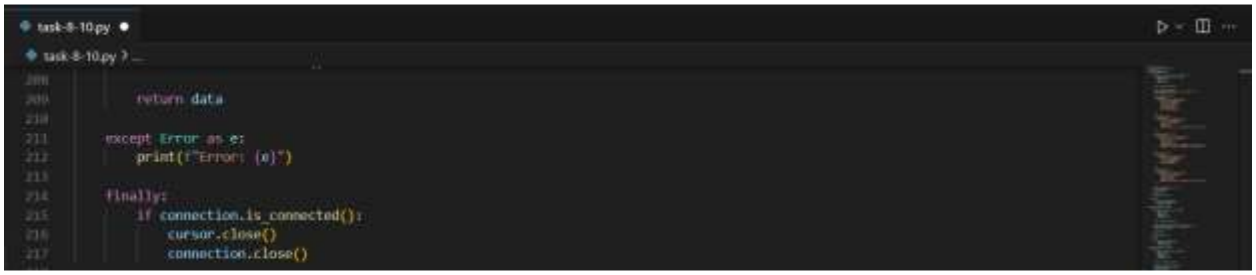
```python
        # Example: Retrieve all data from a specific table
        cursor.execute(f"SELECT * FROM {table_name}")
        data = cursor.fetchall()

        return data

    except Error as e:
        print(f"Error: {e}")

    finally:
        if connection.is_connected():
            cursor.close()
            connection.close()

def insert_data(query, values):
    try:
        connection = mysql.connector.connect(
            host='localhost',
            user='root',
            password='root',
            database='sisdb'
        )

        cursor = connection.cursor()

        # Example: Insert data into a table
        cursor.execute(query, values)
        connection.commit()

    except Error as e:
        print(f"Error: {e}")

    finally:
        if connection.is_connected():
            cursor.close()
            connection.close()

```


task-8-10.py
task-8-10.py > initialize_database

```python
def update_data(query, values):
    try:
        connection = mysql.connector.connect(
            host='localhost',
            user='root',
            password='root',
            database='sisdb'
        )

        cursor = connection.cursor()

        # Example: Update data in a table
        cursor.execute(query, values)
        connection.commit()

    except Error as e:
        print(f"Error: {e}")

    finally:
        if connection.is_connected():
            cursor.close()
            connection.close()

def manage_transaction(queries, values):
    try:
        connection = mysql.connector.connect(
            host='localhost',
            user='root',
            password='root',
            database='sisdb'
        )

        cursor = connection.cursor()

        # Example: Transaction management
        connection.start_transaction()
```

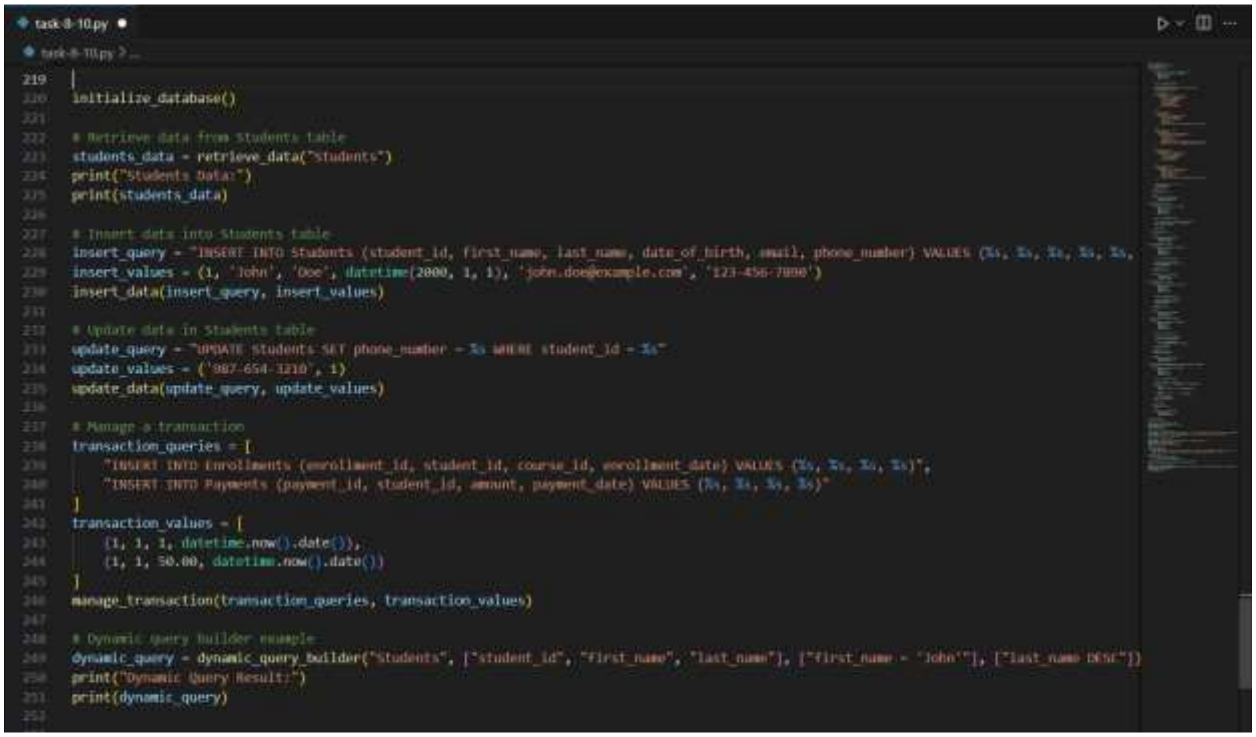

task-8-10.py
task-8-10.py > initialize_database

```python
        for i in range(len(queries)):
            cursor.execute(queries[i], values[i])

        connection.commit()

    except Error as e:
        print(f"Error: {e}")
        connection.rollback()

    finally:
        if connection.is_connected():
            cursor.close()
            connection.close()

def dynamic_query_builder(table_name, columns, conditions, sorting):
    try:
        connection = mysql.connector.connect(
            host='localhost',
            user='root',
            password='root',
            database='sisdb'
        )

        cursor = connection.cursor()

        # Example: Dynamic query builder
        query = f"SELECT {', '.join(columns)} FROM {table_name}"

        if conditions:
            query += f" WHERE {' AND '.join(conditions)}"

        if sorting:
            query += f" ORDER BY {', '.join(sorting)}"

        cursor.execute(query)
        data = cursor.fetchall()
```



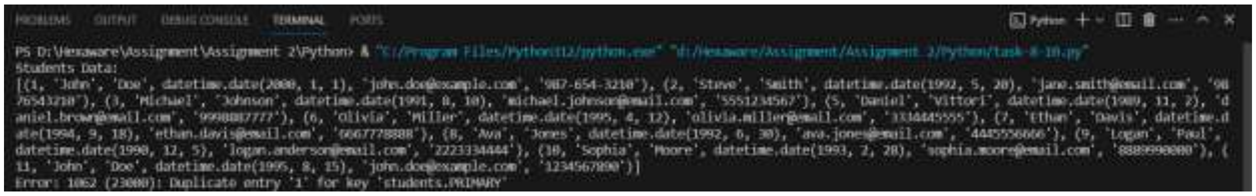
```python
            return data

    except Error as e:
        print(f"Error: {e}")

    finally:
        if connection.is_connected():
            cursor.close()
            connection.close()
```

Inputs for database queries:

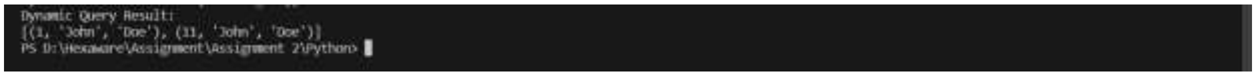


```python

initialize_database()

# Retrieve data from Students table
students_data = retrieve_data("students")
print("Students Data:")
print(students_data)

# Insert data into Students table
insert_query = "INSERT INTO Students (student_id, first_name, last_name, date_of_birth, email, phone_number) VALUES (%s, %s, %s, %s, %s,
insert_values = (1, 'John', 'Doe', datetime(2000, 1, 1), 'john.doe@example.com', '123-456-7890')
insert_data(insert_query, insert_values)

# Update data in Students table
update_query = "UPDATE students SET phone_number = %s WHERE student_id = %s"
update_values = ('987-654-3210', 1)
update_data(update_query, update_values)

# Manage a transaction
transaction_queries = [
    "INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date) VALUES (%s, %s, %s, %s)",
    "INSERT INTO Payments (payment_id, student_id, amount, payment_date) VALUES (%s, %s, %s, %s)"
]
transaction_values = [
    (1, 1, 1, datetime.now().date()),
    (1, 1, 50.00, datetime.now().date())
]
manage_transaction(transaction_queries, transaction_values)

# Dynamic query builder example
dynamic_query = dynamic_query_builder("Students", ["student_id", "first_name", "last_name"], ["first_name = 'John'"], ["last_name DESC"])
print("Dynamic Query Result:")
print(dynamic_query)
```

Execution:

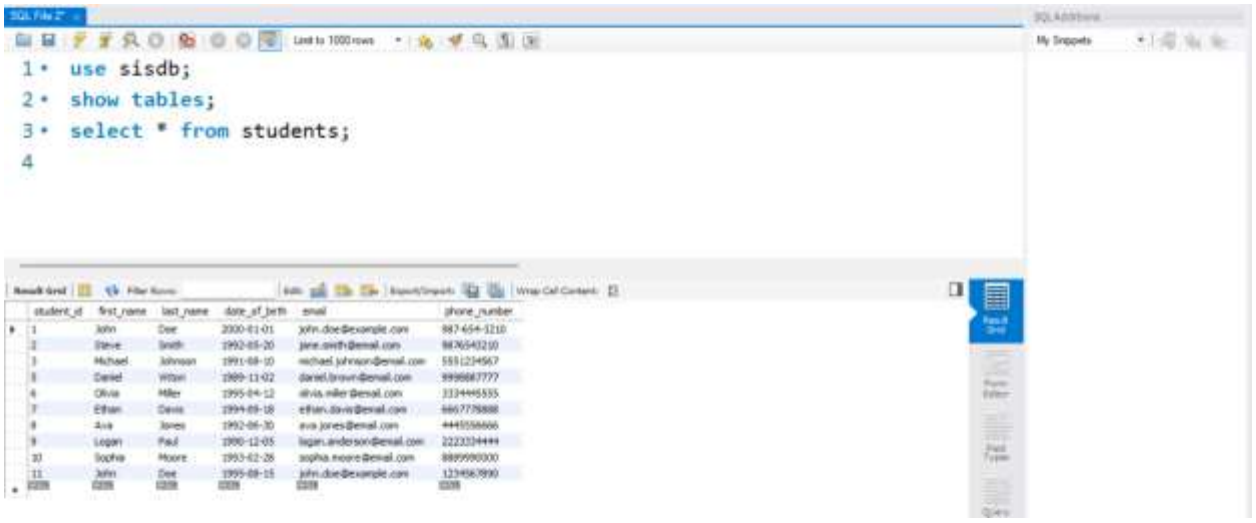


```
PS D:\Hexaware\Assignment\Assignment 2\Python> & "C:/Program Files/Python312/python.exe" "d:/Hexaware/Assignment/Assignment 2/Python/task-8-10.py"
Students Data:
[(1, 'John', 'Doe', datetime.date(2000, 1, 1), 'john.doe@example.com', '987-654-3210'), (2, 'Steve', 'Smith', datetime.date(1992, 5, 20), 'jane.smith@email.com', '98
76543210'), (3, 'Michael', 'Johnson', datetime.date(1991, 8, 10), 'michael.johnson@email.com', '5551234567'), (5, 'Daniel', 'Vittori', datetime.date(1989, 11, 2), 'd
aniel.brown@email.com', '9998887777'), (6, 'Olivia', 'Miller', datetime.date(1995, 4, 12), 'olivia.miller@email.com', '3334445555'), (7, 'Ethan', 'Davis', datetime.d
ate(1994, 9, 18), 'ethan.davis@email.com', '6667778888'), (8, 'Ava', 'Jones', datetime.date(1992, 6, 30), 'ava.jones@email.com', '4445556666'), (9, 'Logan', 'Paul',
datetime.date(1990, 12, 5), 'logan.anderson@email.com', '2223334444'), (10, 'Sophia', 'Moore', datetime.date(1993, 2, 28), 'sophia.moore@email.com', '8889990000'), (
11, 'John', 'Doe', datetime.date(1995, 8, 15), 'john.doe@example.com', '1234567890')]
Error: 1062 (23000): Duplicate entry '1' for key 'students.PRIMARY'
```



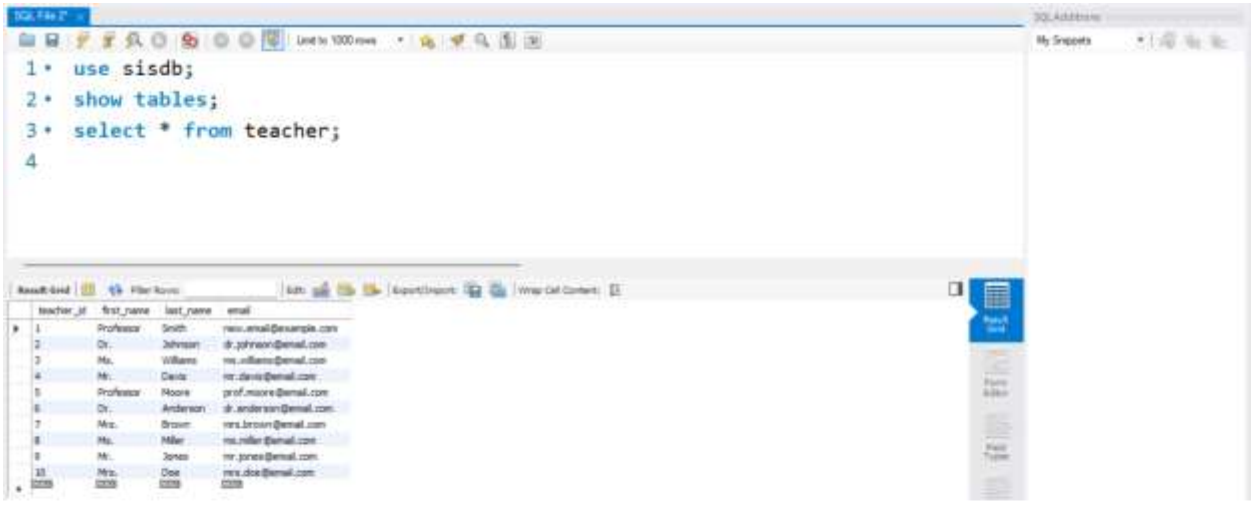
```
Dynamic Query Result:
[(1, 'John', 'Doe'), (11, 'John', 'Doe')]
PS D:\Hexaware\Assignment\Assignment 2\Python>
```

Database Changes:



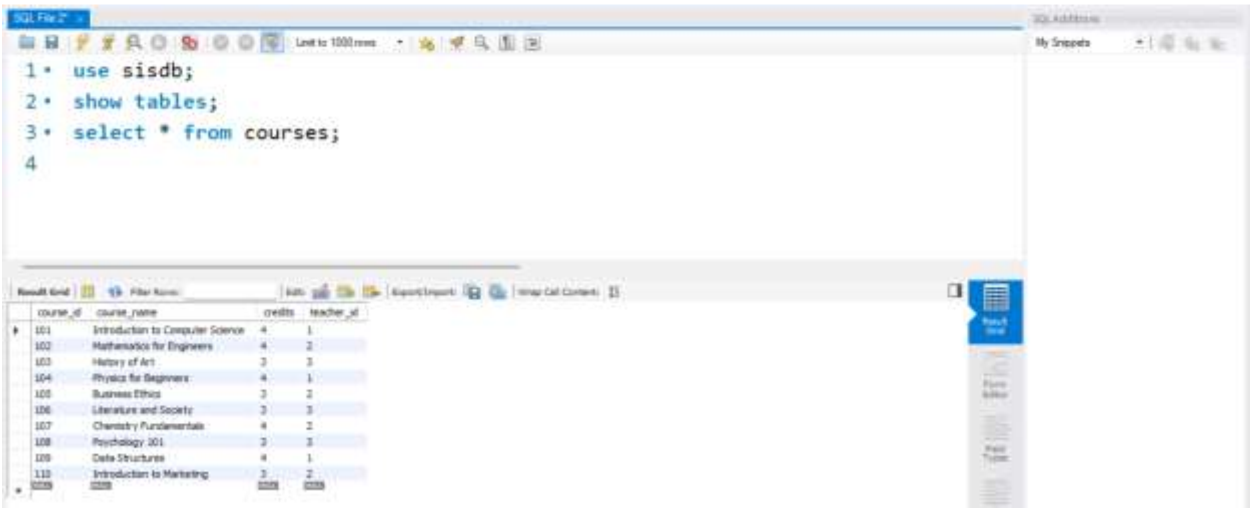
```
use sisdb;
show tables;
select * from students;
```

student_id	first_name	last_name	date_of_birth	email	phone_number
1	John	Doe	2000-01-01	john.doe@example.com	987-654-3210
2	Steve	Smith	1992-05-20	jane.smith@email.com	9876543210
3	Michael	Johnson	1991-08-10	michael.johnson@email.com	5551234567
5	Daniel	Wilson	1989-11-02	daniel.brown@email.com	9998887777
6	Olivia	Miller	1995-04-12	olivia.miller@email.com	3334445555
7	Ethan	Davis	1994-09-18	ethan.davis@email.com	6667778888
8	Ava	Jones	1992-06-30	ava.jones@email.com	4445556666
9	Logan	Paul	1990-12-05	logan.anderson@email.com	2223334444
10	Sophia	Moore	1993-02-28	sophia.moore@email.com	8889990000
11	John	Doe	1995-08-15	john.doe@example.com	1234567890
NULL	NULL	NULL	NULL	NULL	NULL



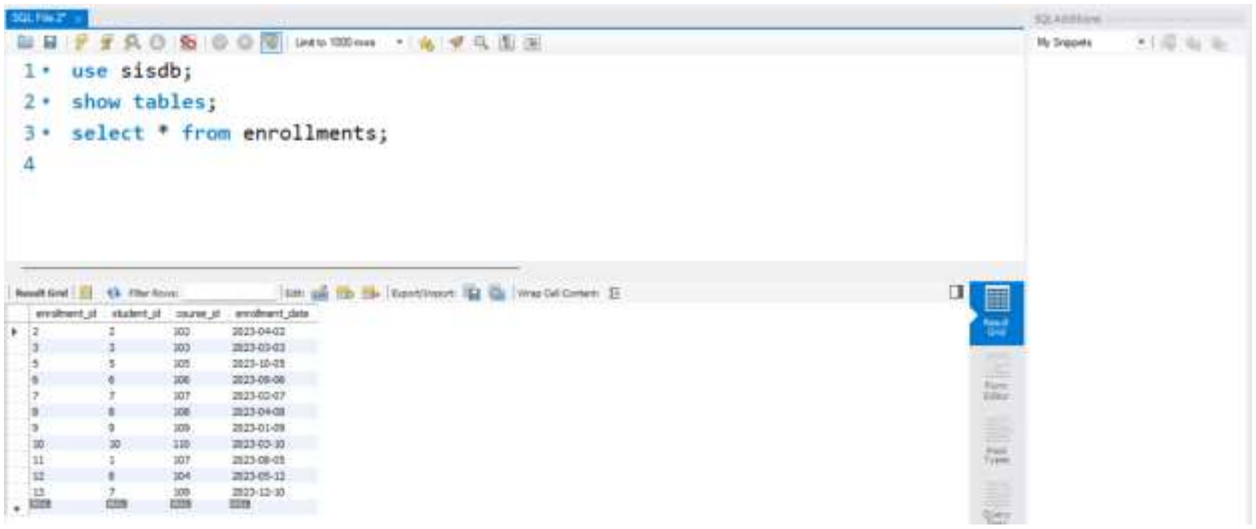
```
use sisdb;
show tables;
select * from teacher;
```

teacher_id	first_name	last_name	email
1	Professor	Smith	new.email@example.com
2	Dr.	Johnson	dr.johnson@email.com
3	Ms.	Williams	ms.williams@email.com
4	Mr.	Davis	mr.davis@email.com
5	Professor	Moore	prof.moore@email.com
6	Dr.	Anderson	dr.anderson@email.com
7	Mrs.	Brown	mrs.brown@email.com
8	Ms.	Miller	ms.miller@email.com
9	Mr.	Jones	mr.jones@email.com
10	Mrs.	Doe	mrs.doe@email.com
NULL	NULL	NULL	NULL



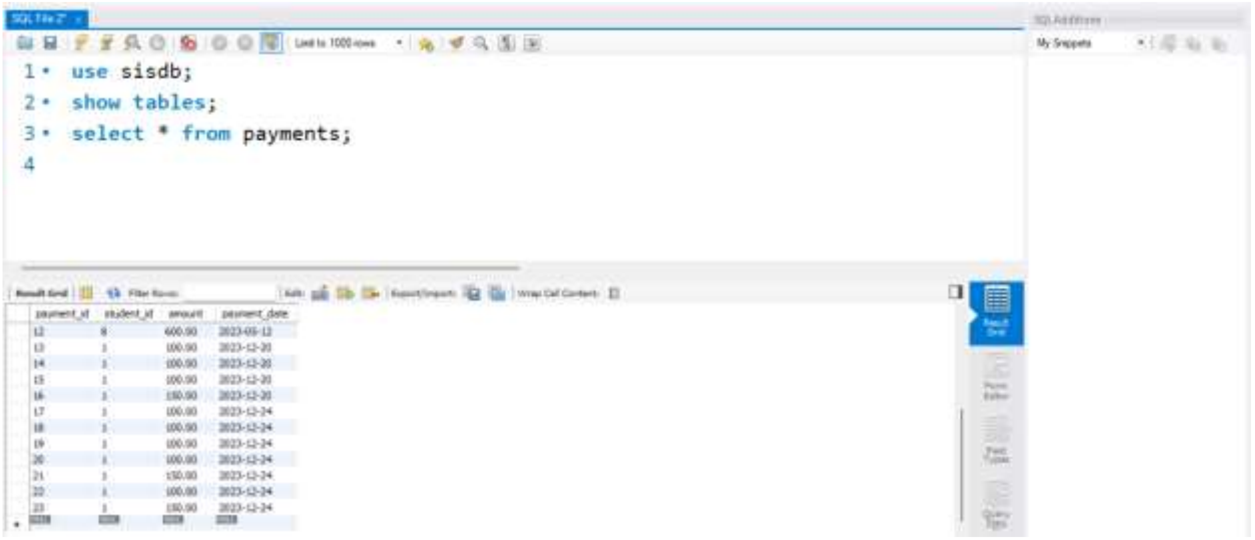
```sql
use sisdb;
show tables;
select * from courses;
```

course_id	course_name	credits	teacher_id
101	Introduction to Computer Science	4	1
102	Mathematics for Engineers	4	2
103	History of Art	3	3
104	Physics for Beginners	4	1
105	Business Ethics	3	2
106	Literature and Society	3	3
107	Chemistry Fundamentals	4	2
108	Psychology 101	3	3
109	Data Structures	4	1
110	Introduction to Marketing	3	2
NULL	NULL	NULL	NULL



```sql
use sisdb;
show tables;
select * from enrollments;
```

enrollment_id	student_id	course_id	enrollment_date
2	2	102	2023-04-02
3	3	103	2023-03-03
5	5	105	2023-10-05
6	6	106	2023-09-06
7	7	107	2023-02-07
8	8	108	2023-04-08
9	9	109	2023-01-09
10	10	110	2023-03-10
11	1	107	2023-08-05
12	8	104	2023-05-12
13	7	109	2023-12-30
NULL	NULL	NULL	NULL



```sql
use sisdb;
show tables;
select * from payments;
```

payment_id	student_id	amount	payment_date
12	8	600.00	2023-05-12
13	1	100.00	2023-12-20
14	1	100.00	2023-12-20
15	1	100.00	2023-12-20
16	1	150.00	2023-12-20
17	1	100.00	2023-12-24
18	1	100.00	2023-12-24
19	1	100.00	2023-12-24
20	1	100.00	2023-12-24
21	1	150.00	2023-12-24
22	1	100.00	2023-12-24
23	1	150.00	2023-12-24
NULL	NULL	NULL	NULL