Student Info System Python Assignment

Task 1 & 2 Defining classes and constructors:

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

```
class Student:
    def
__init__(self,student_id,first_name,last_name,DOB,email,phone_number):
        self.student_id=student_id
        self.first_name=first_name
        self.last_name=last_name
        self.DOB=DOB
        self.email=email
        self.phone_number=phone_number
        self.enrolled_courses=[]
        self.payment_history=[]
```

Course class with the following attributes:

• Course ID

• Course Name

• Course Code

• Instructor Name

```
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.enrolled_students=[]
        self.instructor_list=[]
        self.instructor_list.append(Instructor_name)
```

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

```
class Enrollment(Student,Course):
    def __init__(self,enrollment_id,enrollment_date):
        self.enrollment_id=enrollment_id
        self.student_id=Student.student_id
        self.course_id=Course.course_id
        self.enrollment_date=enrollment_date
```

Teacher class with the following attributes:

• Teacher ID

• First Name

• Last Name

• Email

```
class Teacher(Course):
    def __init__(self,teacher_id,first_name,last_name,email):
        self.teacher_id=self.teacher_id
        self.first_name=self.first_name
        self.last_name=self.last_name
        self.email=self.email
        self.assigned_courses=[]
```

Payment class with the following attributes:

• Payment ID

• Student ID (reference to a Student)

• Amount

• Payment Date

```
class Payment(Student):
    def __init__(self,payment_id,amount,payment_date):
        self.payment_id=payment_id
        self.student_id=Student.student_id
        self.amount=amount
        self.payment_date=payment_date
```

class SIS:

```
class SIS:
    def enrollStudentsInCourse(self,Student,Course):
        self.student_id=Student.student_id
        self.course_id=Course.course_id
        self.student_fname=student.first_name
        self.student_lname=Student.last_name
        self.course_code=Course.course_code
        self.course_name=Course.course_name
        enrolled_students.append(student_id)
```

Task-3:

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.

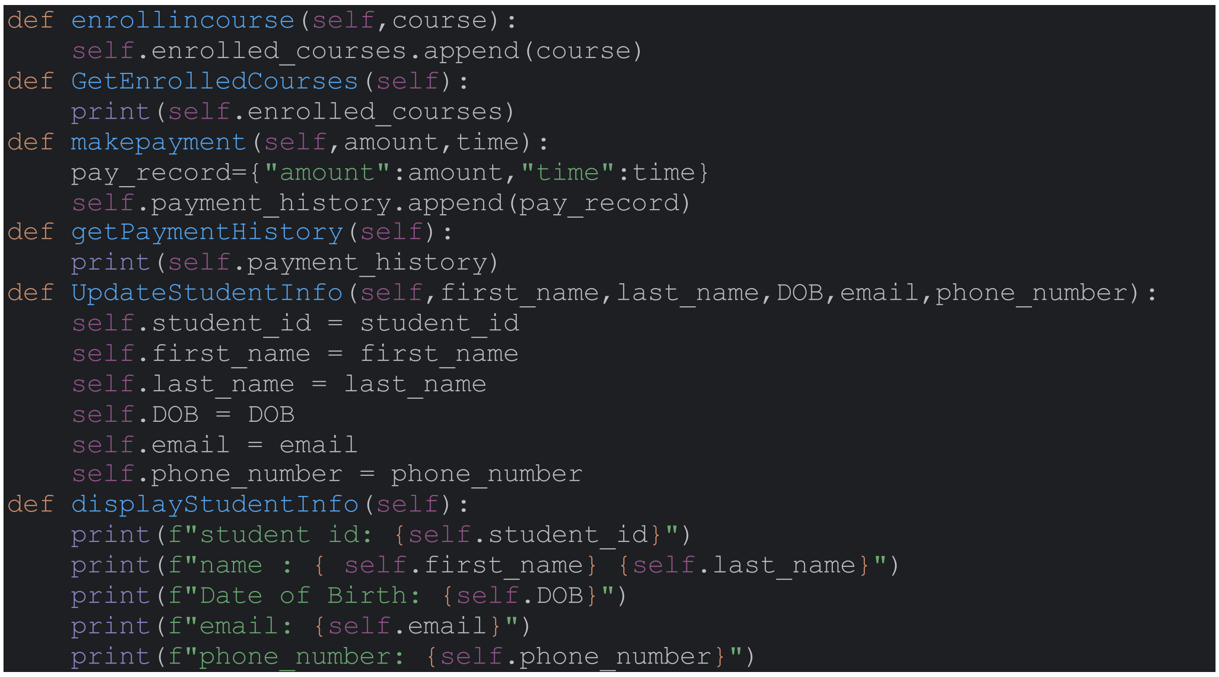
• UpdateStudentInfo(firstName: string, lastName: string, dateOfBirth: DateTime, email: string, phoneNumber: string): Updates the student's information.

• 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.



```
def enrollincourse(self,course):
    self.enrolled_courses.append(course)
def GetEnrolledCourses(self):
    print(self.enrolled_courses)
def makepayment(self,amount,time):
    pay_record={"amount":amount,"time":time}
    self.payment_history.append(pay_record)
def getPaymentHistory(self):
    print(self.payment_history)
def UpdateStudentInfo(self,first_name,last_name,DOB,email,phone_number):
    self.student_id = student_id
    self.first_name = first_name
    self.last_name = last_name
    self.DOB = DOB
    self.email = email
    self.phone_number = phone_number
def displayStudentInfo(self):
    print(f"student id: {self.student_id}")
    print(f"name : { self.first_name} {self.last_name}")
    print(f"Date of Birth: {self.DOB}")
    print(f"email: {self.email}")
    print(f"phone_number: {self.phone_number}")
```

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.

```
def assignTeacher(self,teacher):
    self.instructor_name=teacher
    self.instructor_list.append(teacher)
def updateCourseInfo(self,courseCode,courseName,instructor):
    self.course_code=courseCode
    self.course_name=courseName
    self.instructor=instructor
    enrolled_students.append(self.student_id)
def displayCourseInfo(self):
    print(f"course code: {self.course_code}")
    print(f"course name: {self.course_name}")
    print(f"instructor : {self.Instructor_name}")
def getEnrollments(self):
    print(self.enrolled_students)
def getTeacher(self):
    print(self.instructor_list)
```

Enrollment Class:

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

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

```
def getStudent(self):
    print(self.student_id)
def getCourse(self):
    print(self.course_id)
```

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.

```
def updateTeacherInfo(self,teacher_id,first_name,last_name,email):
    self.teacher_id = self.teacher_id
    self.first_name = self.first_name
    self.last_name = self.last_name
    self.email = self.email
def displayTeacherInfo(self):
    print(f"teacher id: {self.teacher_id}")
    print(f"name: {self.first_name} {self.last_name}")
    print(f"email:{self.email} ")
def assign_course(self,course):
    self.course=course
    assigned_courses.append(course)
def getAssignedCourses(self):
    print(self.assigned_courses)
```

Payment Class:

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

• GetPaymentAmount(): Retrieves the payment amount.

• GetPaymentDate(): Retrieves the payment date.

```
def getstudent(self):
    print(self.student_id)
def getAmount(self):
    print(self.amount)
def getPaymentDate(self):
    print(self.payment_date)
```

SIS Class (if you have one to manage interactions):

• EnrollStudentInCourse(student: Student, course: Course): Enrolls a student in a course.

• AssignTeacherToCourse(teacher: Teacher, course: Course): Assigns a teacher to a course.

• RecordPayment(student: Student, amount: decimal, paymentDate: DateTime): Records a payment made by a student.

• GenerateEnrollmentReport(course: Course): Generates a report of students enrolled in a specific course.

• GeneratePaymentReport(student: Student): Generates a report of payments made by a specific student.

• CalculateCourseStatistics(course: Course): Calculates statistics for a specific course, such as the number of enrollments and total payments.

```
def assignTeacherToStudent(self,Teacher,student):
    self.student_id=Student.student_id
    self.teacher_id=teacher.teacher_id

def recordPayment(self,Student,amount,payment_date):
    self.student_id=Student.student_id
    self.student_fname=Student.first_name
    self.student_lname=student.last_name
    self.amount=amount
    self.payment_date=payment_date

def generateEnrollmentReport(self,course):
    course.getEnrollments()
def generatePaymentReport(self,Student):
    print(Student.getPaymentHistory())
def calculateCourseStatstics(self,Course,Student):
    print(f"enrolled students for the course: {Course.enrolled_students}")
    k=Student.getPaymentHistory()
    print(f"total payments: {k.keys()}")
```

Task 4: Exceptions handling and Custom Exceptions

• 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.

```
class DuplicateEnrollmentException(Exception):
    def __init__(self, message="Student is already enrolled in this 
course"):
        self.message = message
        super().__init__(self.message)
```

• CourseNotFoundException: Thrown when a course does not exist in the system, and you

attempt to perform operations on it (e.g., enrolling a student or assigning a teacher).

```
class CourseNotFoundException(Exception):
    def __init__(self, message="Course not found"):
        self.message = message
        super().__init__(self.message)
```

• StudentNotFoundException: Thrown when a student does not exist in the system, and you

attempt to perform operations on the student (e.g., enrolling in a course, making a payment).

```
class StudentNotFoundException(Exception):
    def __init__(self, message="Student not found"):
        self.message = message
        super().__init__(self.message)
```

• TeacherNotFoundException: Thrown when a teacher does not exist in the system, and you

attempt to assign them to a course.

```
class TeacherNotFoundException(Exception):
    def __init__(self, message="Teacher not found"):
        self.message = message
        super().__init__(self.message)
```

• PaymentValidationException: Thrown when there is an issue with payment validation, such as an invalid payment amount or payment date.

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

• InvalidStudentDataException: Thrown when data provided for creating or updating a student is invalid (e.g., invalid date of birth or email format).

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

• InvalidCourseDataException: Thrown when data provided for creating or updating a course is invalid (e.g., invalid course code or instructor name).

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

• InvalidEnrollmentDataException: Thrown when data provided for creating an enrollment is invalid (e.g., missing student or course references).

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

• InvalidTeacherDataException: Thrown when data provided for creating or updating a teacher is invalid (e.g., missing name or email).

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

• InsufficientFundsException: Thrown when a student attempts to enroll in a course but does not have enough funds to make the payment.

```
class InsufficientFundsException(Exception):
    def __init__(self, message="Insufficient funds"):
        self.message = message
        super().__init__(self.message)
```

task-5 is implemented by taking extra data structures to store the information in every class

`self.enrollments = []` In student Class

`self.assigned_courses = []` In Teacher class

Task-6

Define Class-Level Data Structures

You will need class-level data structures within each class to maintain relationships. Here's how to

define them for each class:

Student Class:

Create a list or collection property to store the student's enrollments. This property will hold references

to Enrollment objects.

Example: List<Enrollment> Enrollments { get; set; }

Course Class:

Create a list or collection property to store the course's enrollments. This property will hold references

to Enrollment objects.

Example: List<Enrollment> Enrollments { get; set; }

Enrollment Class:

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

Example: Student Student { get; set; } and Course Course { get; set; }

Teacher Class:

Create a list or collection property to store the teacher's assigned courses. This property will hold

references to Course objects

We need To create them in SIS as it is used for managing the classes

```python
def add_enrollment(self, student, course, enrollment_date):
    enrollment = Enrollment(enrollment_id, student, course,
enrollment_date)
    student.enrollments.append(enrollment)
    course.enrollments.append(enrollment)

def assign_course_to_teacher(self, course, teacher):
    teacher.assigned_courses.append(course)

def add_payment(self, student, amount, payment_date):
    payment = Payment(payment_id, student, amount, payment_date)
    student.payments.append(payment)

def get_enrollments_for_student(self, student):
    return student.enrollments

def get_courses_for_teacher(self, teacher):
    return teacher.assigned_courses
```

Task-7

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.

Data Retrieval:

Implement methods to retrieve data from the database. Users should be able to request information

about students, courses, enrollments, teachers, or payments. Ensure that the data retrieval methods

handle exceptions and edge cases gracefully.

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.

Transaction Management:

Implement methods for handling database transactions when enrolling students, assigning teachers, or

recording payments. Transactions should be atomic and maintain data integrity. Use database

transactions to ensure that multiple related operations either all succeed or all fail. Implement error

handling and rollback mechanisms in case of transaction failures.

Dynamic Query Builder:

Implement a dynamic query builder that allows users to construct and execute custom SQL queries to

retrieve specific data from the database. Users should be able to specify columns, conditions, and

sorting criteria. Create a query builder method that dynamically generates SQL queries based on user

input. Implement parameterization and sanitation of user inputs to prevent SQL injection.

```
db=mysql.connector.connect(
    host="localhost",
    user="root",
    password= "Vishnu@542",
    database="SISD"
)
```

Data retrival:

```
def retrieve_data(self,table_name):
    cursor=database.db.cursor()
    query=f"select * from {table_name}"
    cursor.execute(query)
    answer=cursor.fetchall()
    for x in answer:
        print(x)
    cursor.close()
```

data insertion:

```
def
insert_data_students(self,student_id,first_name,last_name,date
_of_birth,email,phone_number):
    cursor=database.db.cursor()
    query=f"insert into students values(%s,%s,%s,%s,%s,%s)"

data=(student_id,first_name,last_name,date_of_birth,email,phon
e_number)
    cursor.execute(query, data)
    print(f"data entered into students table")
    cursor.commit()
    cursor.close()

def insert_data_courses(self,
course_id,course_name,credits,teacher_id):
    cursor = database.db.cursor()
    query = f"insert into courses values(%s,%s,%s,%s)"
    data = (course_id,course_name,credits,teacher_id)
    cursor.execute(query, data)
    print(f"data entered into courses table")
    cursor.commit()
    cursor.close()

def
insert_data_teacher(self,teacher_id,first_name,last_name,email
):
    cursor = database.db.cursor()
    query = f"insert into teacher values(%s,%s,%s,%s)"
    data = (teacher_id,first_name,last_name,email)
    cursor.execute(query, data)
    print(f"data entered into teacher table")
    cursor.commit()
    cursor.close()
```

```
def
insert_data_enrollments(self,enrollment_id,student_id,course_i
d,enrollment_date):
    cursor = database.db.cursor()
    query = f"insert into enrollments values(%s,%s,%s,%s)"
    data =
(enrollment_id,student_id,course_id,enrollment_date)
    cursor.execute(query, data)
    print(f"data entered into enrollments table")
    cursor.commit()
    cursor.close()
```

Table Updation:

```
def update_data(self, table_name,column_name,id,value):
   if(table_name='students'):
        identifier='student_id'
   elif(table_name='courses'):
       identifier='course_id'
   elif(table_name='enrollments'):
       identifier='enrollment_id'
   elif(table_name="payments"):
       identifier='payment_id'
   else:
       identifier='teacher_id'
    cursor = database.db.cursor()
    query=f"update {table_name} set {column_name}={value}
where {identifier}={id}"
    cursor.execute(query)
    print(f"updated {table_name}")
    cursor.commit()
    cursor.close()
```

Task 8:

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

John is enrolling in the following courses:

• Course 1: Introduction to Programming

• Course 2: Mathematics 101

The system should perform the following tasks:

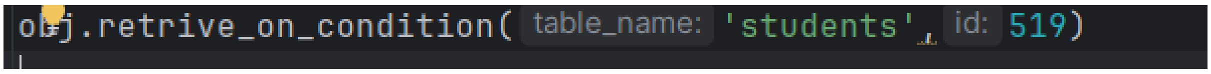
• Create a new student record in the database.

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

Include the student using the following way

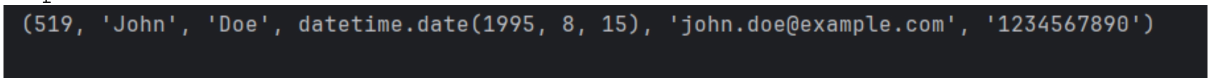
```
def
insert_data_students(self,student_id,first_name,last_name,date
_of_birth,email,phone_number):
    cursor=database.db.cursor()
    query=f"insert into students values(%s,%s,%s,%s,%s,%s)"

data=(student_id,first_name,last_name,date_of_birth,email,phon
e_number)
    cursor.execute(query, data)
    print(f"data entered into students table")
    cursor.commit()
    cursor.close()
```



```
obj.retrive_on_condition( table_name: 'students', id: 519)
```

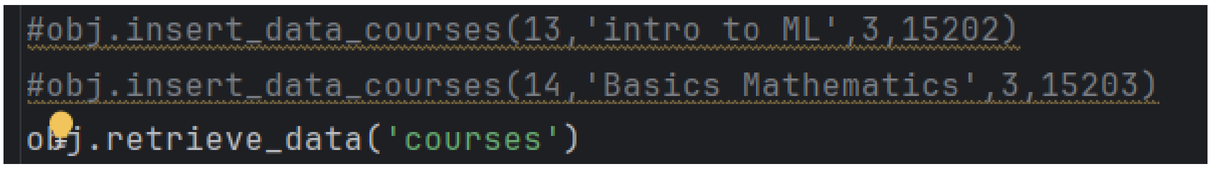
o/p:



```
(519, 'John', 'Doe', datetime.date(1995, 8, 15), 'john.doe@example.com', '1234567890')
```

now add him to course using following method:

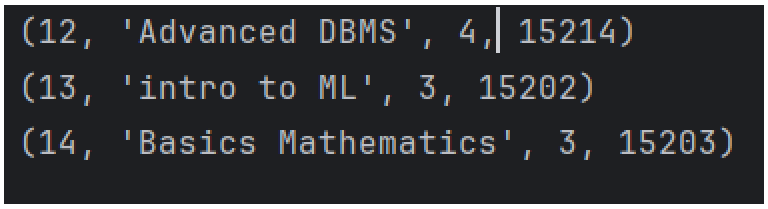
```
def add_to_course(self,student_name,course_name):
    cursor=database.db.cursor()
    q1=f"select student_id from students where
first_name={student_name} or last_name={student_name}"
    cursor.execute(q1)
    stid=cursor.fetch()
    q2=f"select course_id from courses where
course_name={course_name} "
    cursor.execute(q2)
    cid=cursor.fetch()
    SIS.insert_data_enrollments(stid,cid,datetime.now())
    cursor.commit()
    cursor.close()
```



```
#obj.insert_data_courses(13,'intro to ML',3,15202)
#obj.insert_data_courses(14,'Basics Mathematics',3,15203)
obj.retrieve_data('courses')
```

Courses added to database

o/p:



```
(12, 'Advanced DBMS', 4, 15214)
(13, 'intro to ML', 3, 15202)
(14, 'Basics Mathematics', 3, 15203)
```

Now adding the student to course

task-9:

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

Course to be assigned:

• Course Name: Advanced Database Management

• Course Code: CS302

The system should perform the following tasks:

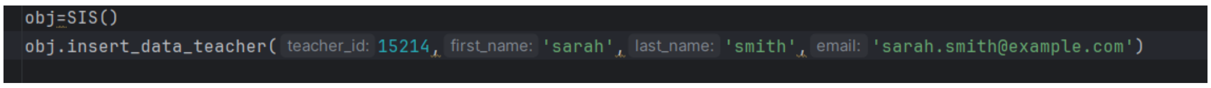
• Retrieve the course record from the database based on the course code.

• Assign Sarah Smith as the instructor for the course.

• Update the course record in the database with the new instructor information.

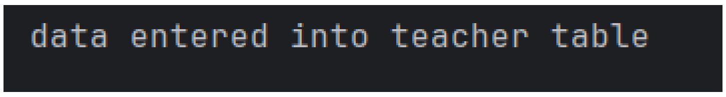
Insert the new teacher into the teachers by using following method:

```
def
insert_data_teacher(self,teacher_id,first_name,last_name,email
):
    cursor = database.db.cursor()
    query = f"insert into teacher values(%s,%s,%s,%s)"
    data = (teacher_id,first_name,last_name,email)
    cursor.execute(query, data)
    print(f"data entered into teacher table")
    cursor.commit()
    cursor.close()
```



```
obj=SIS()
obj.insert_data_teacher( teacher_id: 15214, first_name: 'sarah', last_name: 'smith', email: 'sarah.smith@example.com')
```

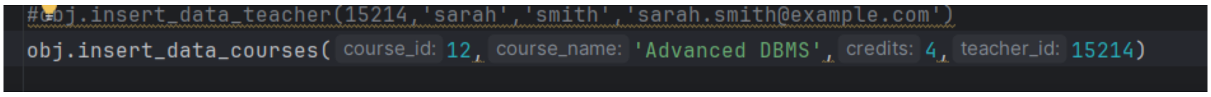
o/p:



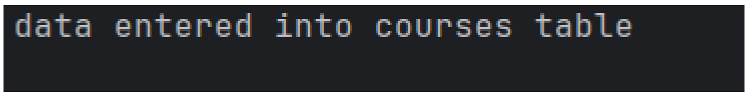
```
data entered into teacher table
```

Now create the courses into courses table:

```
def insert_data_courses(self,course_name,credits,teacher_id):
    cursor = database.db.cursor()
    query = f"insert into courses values(%s,%s,%s)"
    data = (course_id,course_name,credits,teacher_id)
    cursor.execute(query, data)
    print(f"data entered into courses table")
    cursor.commit()
    cursor.close()
```



```
#obj.insert_data_teacher(15214,'sarah','smith','sarah.smith@example.com')
obj.insert_data_courses( course_id: 12, course_name: 'Advanced DBMS', credits: 4, teacher_id: 15214)
```



```
data entered into courses table
```

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

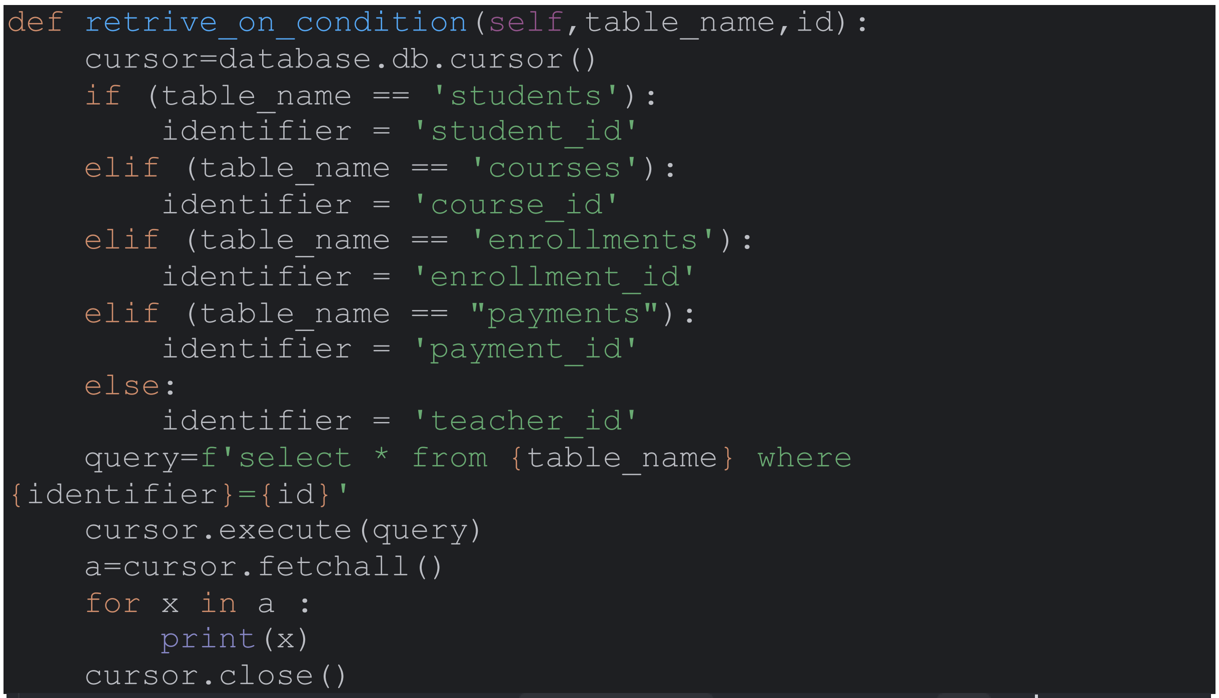
The system should perform the following tasks:

• Retrieve Jane Johnson's student record from the database based on her student ID.

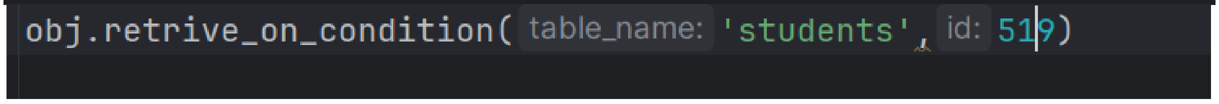
• Record the payment information in the database, associating it with Jane's student record.

• Update Jane's outstanding balance in the database based on the payment amount.

Retriving the student record:

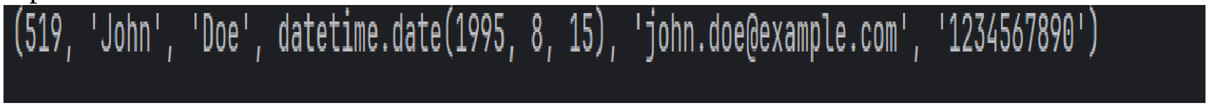


```
def retrive_on_condition(self,table_name,id):
    cursor=database.db.cursor()
    if (table_name == 'students'):
        identifier = 'student_id'
    elif (table_name == 'courses'):
        identifier = 'course_id'
    elif (table_name == 'enrollments'):
        identifier = 'enrollment_id'
    elif (table_name == "payments"):
        identifier = 'payment_id'
    else:
        identifier = 'teacher_id'
    query=f'select * from {table_name} where
{identifier}={id}'
    cursor.execute(query)
    a=cursor.fetchall()
    for x in a :
        print(x)
    cursor.close()
```



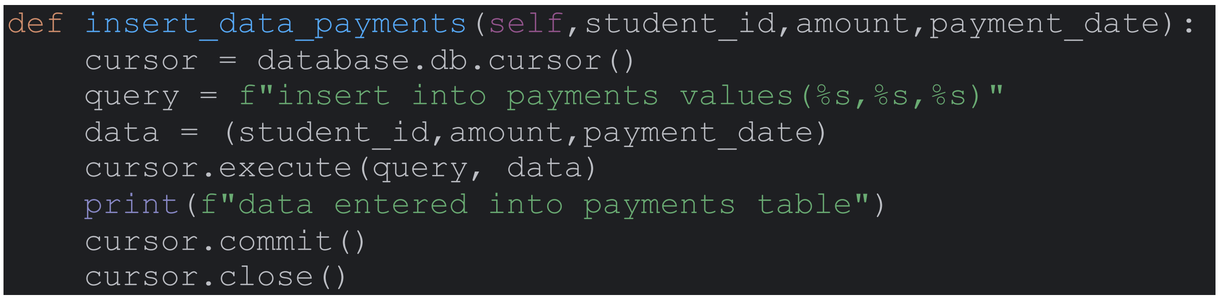
```
obj.retrive_on_condition( table_name: 'students', id: 519)
```

o/p:



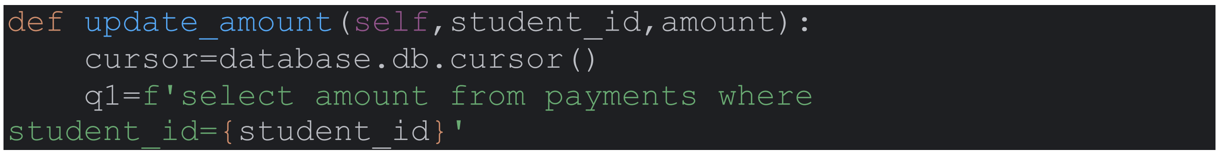
```
(519, 'John', 'Doe', datetime.date(1995, 8, 15), 'john.doe@example.com', '1234567890')
```

storing the record in database:



```
def insert_data_payments(self,student_id,amount,payment_date):
    cursor = database.db.cursor()
    query = f"insert into payments values(%s,%s,%s)"
    data = (student_id,amount,payment_date)
    cursor.execute(query, data)
    print(f"data entered into payments table")
    cursor.commit()
    cursor.close()
```

updating the amount:



```
def update_amount(self,student_id,amount):
    cursor=database.db.cursor()
    q1=f'select amount from payments where
student_id={student_id}'
```

```
    cursor.execute(q1)
    a=cursor.fetch()+amount
    query=f'update payments set amount={a}'
    cursor.execute(query)
    cursor.commit()
    cursor.close()
    print("amount updated")
```

task 11 :

generating enrolment report:

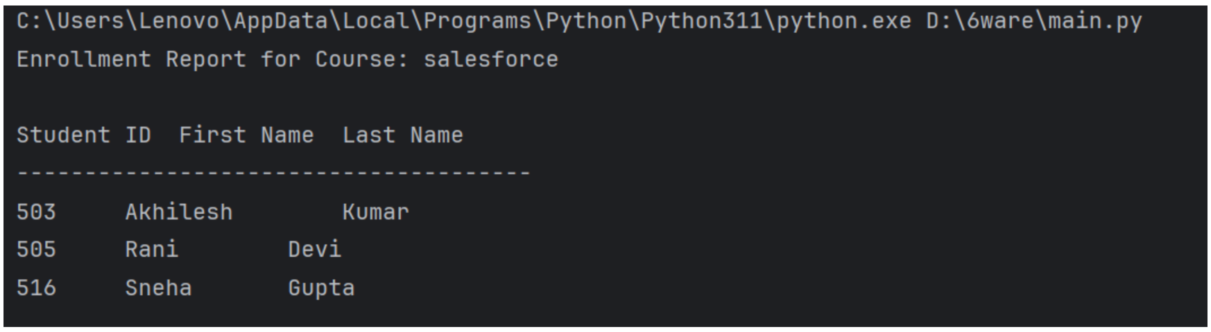
```
def generate_enrollment_report(self, course_name):

        cursor = database.db.cursor()
        query = ("SELECT students.student_id,
students.first_name, students.last_name "
                 "FROM enrollments "
                 "JOIN students ON enrollments.student_id =
students.student_id "
                 "JOIN courses ON enrollments.course_id =
courses.course_id "
                 "WHERE courses.course_name = %s")
        cursor.execute(query, (course_name,))
        enrollments = cursor.fetchall()

        if enrollments:
            print(f"Enrollment Report for Course:
{course_name}\n")
            print("Student ID\tFirst Name\tLast Name")
            print("---------------------------------------")
            for enrollment in enrollments:
                student_id, first_name, last_name = enrollment

print(f"{student_id}\t\t{first_name}\t\t{last_name}")
        else:
            print(f"No enrollments found for course:
{course_name}")
```

o/p:



```
C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\python.exe D:\6ware\main.py
Enrollment Report for Course: salesforce

Student ID  First Name  Last Name
---------------------------------------
503     Akhilesh        Kumar
505     Rani        Devi
516     Sneha       Gupta
```