

Assignment 2

Student Information System

Task(1,2,3):

```
course.py X
course.py > Course
1 import mysql.connector
2 from datetime import datetime
3 from exceptions import TeacherNotFoundException, InvalidCourseDataException
4
5 class Course:
6     def __init__(self, course_id, course_name, credits, teacher_id):
7         self.course_id = course_id
8         self.course_name = course_name
9         self.credits = credits
10        self.teacher_id = teacher_id
11        self.conn = mysql.connector.connect(user='root', password='root', host='localhost', database='sisdh')
12        self.cursor = self.conn.cursor()
13
14        def assign_teacher(self, teacher_id):
15            # Check if the teacher exists
16            query = "SELECT * FROM Teacher WHERE teacher_id=%s"
17            self.cursor.execute(query, (teacher_id,))
18            if not self.cursor.fetchone():
19                raise TeacherNotFoundException(teacher_id)
20
21            # Assign a teacher to the course
22            query = "UPDATE Courses SET teacher_id=%s WHERE course_id=%s"
23            values = (teacher_id, self.course_id)
24            self.cursor.execute(query, values)
25            self.conn.commit()
26
27        def update_course_info(self, course_name, credits):
28            if not course_name or not credits:
29                raise InvalidCourseDataException("Invalid course data")
30
31            # Update course information
32            query = "UPDATE Courses SET course_name=%s, credits=%s WHERE course_id=%s"
33            values = (course_name, credits, self.course_id)
34            self.cursor.execute(query, values)
35            self.conn.commit()
36
37        def display_course_info(self):
38            # Display detailed information about the course
39            query = "SELECT * FROM Courses WHERE course_id=%s"
40            self.cursor.execute(query, (self.course_id,))
41            result = self.cursor.fetchone()
42            print("Course Information:")
43            print("Course ID:", result[0])
44            print("Course Name:", result[1])
45            print("Credits:", result[2])
46            print("Teacher ID:", result[3])
47
48        def get_enrollments(self):
49            # Retrieve a list of student enrollments for the course
50            query = "SELECT Students.student_id, First_name, last_name FROM Students JOIN Enrollments ON Students.student_id = Enrollments.st"
51            self.cursor.execute(query, (self.course_id,))
52            enrollments = self.cursor.fetchall()
53            return enrollments
54
55        def get_teacher(self):
56            # Retrieve the assigned teacher for the course
57            query = "SELECT * FROM Teacher WHERE teacher_id=%s"
58            self.cursor.execute(query, (self.teacher_id,))
59            result = self.cursor.fetchone()
60            return result
61
```

```

enrollment.py X
enrollment.py >...
1  import mysql.connector
2
3  class Enrollment:
4      def __init__(self, enrollment_id, student_id, course_id, enrollment_date):
5          self.enrollment_id = enrollment_id
6          self.student_id = student_id
7          self.course_id = course_id
8          self.enrollment_date = enrollment_date
9          self.conn = mysql.connector.connect(user='root', password='root', host='localhost', database='sisdh')
10         self.cursor = self.conn.cursor()
11
12         def get_student(self):
13             # Retrieve the student associated with the enrollment
14             query = "SELECT * FROM Students WHERE student_id=%s"
15             self.cursor.execute(query, (self.student_id,))
16             result = self.cursor.fetchone()
17             return result
18
19         def get_course(self):
20             # Retrieve the course associated with the enrollment
21             query = "SELECT * FROM Courses WHERE course_id=%s"
22             self.cursor.execute(query, (self.course_id,))
23             result = self.cursor.fetchone()
24             return result
25

```

```

payment.py X
payment.py >...
1  import mysql.connector
2  from exceptions import StudentNotFoundException, PaymentValidationException
3  class Payment:
4      def __init__(self, payment_id, student_id, amount, payment_date):
5          self.payment_id = payment_id
6          self.student_id = student_id
7          self.amount = amount
8          self.payment_date = payment_date
9          self.conn = mysql.connector.connect(user='root', password='root', host='localhost', database='sisdh')
10         self.cursor = self.conn.cursor()
11
12         def get_student(self):
13             # Check if the student exists
14             query = "SELECT * FROM Students WHERE student_id=%s"
15             self.cursor.execute(query, (self.student_id,))
16             if not self.cursor.fetchone():
17                 raise StudentNotFoundException(self.student_id)
18
19             # Retrieve the student associated with the payment
20             query = "SELECT * FROM Students WHERE student_id=%s"
21             self.cursor.execute(query, (self.student_id,))
22             result = self.cursor.fetchone()
23             return result
24
25         def get_payment_amount(self):
26             # Validate payment amount
27             if self.amount <= 0:
28                 raise PaymentValidationException("Invalid payment amount")
29
30             # Retrieve the payment amount
31             return self.amount
32
33         def get_payment_date(self):
34             # Retrieve the payment date
35             return self.payment_date
36

```

```
student.py X
student.py Student update_student_info
1 import mysql.connector
2 from datetime import datetime
3 from exceptions import DuplicateEnrollmentException, PaymentValidationException
4
5 class Student:
6     def __init__(self, student_id, first_name, last_name, date_of_birth, email, phone_number):
7         self.student_id = student_id
8         self.first_name = first_name
9         self.last_name = last_name
10        self.date_of_birth = date_of_birth
11        self.email = email
12        self.phone_number = phone_number
13        self.conn = mysql.connector.connect(user='root', password='root', host='localhost', database='sisdb')
14        self.cursor = self.conn.cursor()
15
16    def enroll_in_course(self, course_id):
17        # Check if the student is already enrolled in the course
18        query = "SELECT * FROM Enrollments WHERE student_id=%s AND course_id=%s"
19        self.cursor.execute(query, (self.student_id, course_id))
20        if self.cursor.fetchone():
21            raise DuplicateEnrollmentException(self.student_id, course_id)
22
23        # Enroll the student in a course
24        enrollment_date = datetime.now().date()
25        query = "INSERT INTO Enrollments (student_id, course_id, enrollment_date) VALUES (%s, %s, %s)"
26        values = (self.student_id, course_id, enrollment_date)
27        self.cursor.execute(query, values)
28        self.conn.commit()
29
30    def update_student_info(self, first_name, last_name, date_of_birth, email, phone_number):
31        # Update student information
32
33        query = "UPDATE Students SET first_name=%s, last_name=%s, date_of_birth=%s, email=%s, phone_number=%s WHERE student_id=%s"
34        values = (first_name, last_name, date_of_birth, email, phone_number, self.student_id)
35        self.cursor.execute(query, values)
36        self.conn.commit()
```

```
student.py X
student.py Student update_student_info
37    def make_payment(self, amount):
38        # Record a payment made by the student
39        if amount <= 0:
40            raise PaymentValidationException("Invalid payment amount")
41
42        # Record a payment made by the student
43        payment_date = datetime.now().date()
44        query = "INSERT INTO Payments (student_id, amount, payment_date) VALUES (%s, %s, %s)"
45        values = (self.student_id, amount, payment_date)
46        self.cursor.execute(query, values)
47        self.conn.commit()
48
49    def display_student_info(self):
50        # Display detailed information about the student
51        query = "SELECT * FROM Students WHERE student_id=%s"
52        self.cursor.execute(query, (self.student_id,))
53        result = self.cursor.fetchone()
54        print("Student Information:")
55        print("Student ID:", result[0])
56        print("First Name:", result[1])
57        print("Last Name:", result[2])
58        print("Date of Birth:", result[3])
59        print("Email:", result[4])
60        print("Phone Number:", result[5])
61
62    def get_enrolled_courses(self):
63        # Retrieve a list of courses in which the student is enrolled
64        query = "SELECT Courses.course_id, course_name FROM Courses JOIN Enrollments ON Courses.course_id = Enrollments.course_id WHERE s"
65        self.cursor.execute(query, (self.student_id,))
66        courses = self.cursor.fetchall()
67        return courses
68
69
```

```

63     def get_enrolled_courses(self):
64         # Retrieve a list of courses in which the student is enrolled
65         query = "SELECT Courses.course_id, course_name FROM Courses JOIN Enrollments ON Courses.course_id = Enrollments.course_id WHERE s"
66         self.cursor.execute(query, (self.student_id,))
67         courses = self.cursor.fetchall()
68         return courses
69
70     def get_payment_history(self):
71         # Retrieve a list of payment records for the student
72         query = "SELECT * FROM Payments WHERE student_id=%s"
73         self.cursor.execute(query, (self.student_id,))
74         payments = self.cursor.fetchall()
75         return payments
76
77 sis = Student(1, "John", "Doe", datetime(1995, 8, 15), "john.doe@example.com", "123-456-7890")
78
79 sis.enroll_in_course()
80
81 # Enroll John in the specified courses
82 courses_to_enroll = ["Introduction to Programming", "Mathematics 101"]
83

```

SIS:

```

1  import mysql.connector
2  from datetime import datetime
3  from exceptions import CourseNotFoundException, StudentNotFoundException, DuplicateEnrollmentException, TeacherNotFoundException, PaymentVali
4
5  class SIS:
6      def __init__(self):
7          self.conn = mysql.connector.connect(user='root', password='root', host='localhost', database='sisdb')
8          self.cursor = self.conn.cursor()
9
10     def enroll_student_in_course(self, student_id, course_id):
11         # Check if the course exists
12         query = "SELECT * FROM Courses WHERE course_id=%s"
13         self.cursor.execute(query, (course_id,))
14         if not self.cursor.fetchone():
15             raise CourseNotFoundException(course_id)
16
17         # Check if the student exists
18         query = "SELECT * FROM Students WHERE student_id=%s"
19         self.cursor.execute(query, (student_id,))
20         if not self.cursor.fetchone():
21             raise StudentNotFoundException(student_id)
22
23     def assign_teacher_to_course(self, teacher_id, course_id):
24         # Assign a teacher to a course
25         query = "UPDATE Courses SET teacher_id=%s WHERE course_id=%s"
26         values = (teacher_id, course_id)
27         self.cursor.execute(query, values)
28         self.conn.commit()
29
30     def record_payment(self, student_id, amount):
31         # Record a payment made by a student
32         payment_date = datetime.now().date()
33         query = "INSERT INTO Payments (student_id, amount, payment_date) VALUES (%s, %s, %s)"
34         values = (student_id, amount, payment_date)
35         self.cursor.execute(query, values)
36         self.conn.commit()
37

```



```
student.py  stu.py  X
stu.py > ...
38 def generate_enrollment_report(self, course_id):
39     # Generate a report of students enrolled in a specific course
40     query = "SELECT * FROM Students JOIN Enrollments ON Students.student_id = Enrollments.student_id WHERE course_id=%s"
41     self.cursor.execute(query, (course_id,))
42     report = self.cursor.fetchall()
43     return report
44
45 def generate_payment_report(self, student_id):
46     # Generate a report of payments made by a specific student
47     query = "SELECT * FROM Payments WHERE student_id=%s"
48     self.cursor.execute(query, (student_id,))
49     report = self.cursor.fetchall()
50     return report
51
52 def calculate_course_statistics(self, course_id):
53     # Calculate statistics for a specific course
54     query_enrollments = "SELECT COUNT(*) FROM Enrollments WHERE course_id=%s"
55     query_payments = "SELECT SUM(amount) FROM Payments JOIN Enrollments ON Payments.student_id = Enrollments.student_id WHERE course_id=%s"
56     self.cursor.execute(query_enrollments, (course_id,))
57     num_enrollments = self.cursor.fetchone()[0]
58     self.cursor.execute(query_payments, (course_id,))
59     total_payments = self.cursor.fetchone()[0]
60     return num_enrollments, total_payments
61
62 # Method to add an enrollment to both the student's and course's enrollment lists
63 def add_enrollment(self, student, course, enrollment_date):
64     # Check if the student and course exist
65     query_student = "SELECT * FROM Students WHERE student_id=%s"
66     query_course = "SELECT * FROM Courses WHERE course_id=%s"
67     self.cursor.execute(query_student, (student.student_id,))
68     student_data = self.cursor.fetchone()
69     self.cursor.execute(query_course, (course.course_id,))
70     course_data = self.cursor.fetchone()
71
```

```
student.py  stu.py  X
stu.py > ...
72 if not student_data:
73     raise StudentNotFoundException(student.student_id)
74 if not course_data:
75     raise CourseNotFoundException(course.course_id)
76
77 # Check if the student is already enrolled in the course
78 query_duplicate_enrollment = "SELECT * FROM Enrollments WHERE student_id=%s AND course_id=%s"
79 self.cursor.execute(query_duplicate_enrollment, (student.student_id, course.course_id))
80 if self.cursor.fetchone():
81     raise DuplicateEnrollmentException(student.student_id, course.course_id)
82
83 # Add enrollment to both the student's and course's enrollment lists
84 query_add_enrollment = "INSERT INTO Enrollments (student_id, course_id, enrollment_date) VALUES (%s, %s, %s)"
85 values = (student.student_id, course.course_id, enrollment_date)
86 self.cursor.execute(query_add_enrollment, values)
87 self.conn.commit()
88
89 def assign_course_to_teacher(self, course, teacher):
90     # Check if the course and teacher exist
91     query_course = "SELECT * FROM Courses WHERE course_id=%s"
92     query_teacher = "SELECT * FROM Teacher WHERE teacher_id=%s"
93     self.cursor.execute(query_course, (course.course_id,))
94     course_data = self.cursor.fetchone()
95     self.cursor.execute(query_teacher, (teacher.teacher_id,))
96     teacher_data = self.cursor.fetchone()
97
98 if not course_data:
99     raise CourseNotFoundException(course.course_id)
100 if not teacher_data:
101     raise TeacherNotFoundException(teacher.teacher_id)
102
103 # Assign the course to the teacher
104 query_assign_course = "UPDATE Courses SET teacher_id=%s WHERE course_id=%s"
105 values = (teacher.teacher_id, course.course_id)
106 self.cursor.execute(query_assign_course, values)
107 self.conn.commit()
108
```

```
student.py  student.py  X
spy > _
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)
```

```
student.py  student.py  X
spy > _
145
146 # Retrieve all enrollments for the student
147 query_enrollments = "SELECT * FROM Enrollments WHERE student_id=%s"
148 self.cursor.execute(query_enrollments, (student.student_id,))
149 enrollments = self.cursor.fetchall()
150 return enrollments
151
152 # Method to retrieve all courses assigned to a specific teacher
153 def get_courses_for_teacher(self, teacher):
154     # Check if the teacher exists
155     query_teacher = "SELECT * FROM Teacher WHERE teacher_id=%s"
156     self.cursor.execute(query_teacher, (teacher.teacher_id,))
157     teacher_data = self.cursor.fetchone()
158
159     if not teacher_data:
160         raise TeacherNotFoundException(teacher.teacher_id)
161
162     # Retrieve all courses assigned to the teacher
163     query_courses = "SELECT * FROM Courses WHERE teacher_id=%s"
164     self.cursor.execute(query_courses, (teacher.teacher_id,))
165     courses = self.cursor.fetchall()
166     return courses
167
168
169
```

Taks(4,5,6)Exceptions:

```
exceptions.py X
exceptions.py > _
1 class DuplicateEnrollmentException(Exception):
2     def __init__(self, student_id, course_id):
3         super().__init__(f"Student {student_id} is already enrolled in Course {course_id}")
4
5 class CourseNotFoundException(Exception):
6     def __init__(self, course_id):
7         super().__init__(f"Course {course_id} not found in the system")
8
9 class StudentNotFoundException(Exception):
10     def __init__(self, student_id):
11         super().__init__(f"Student {student_id} not found in the system")
12
13 class TeacherNotFoundException(Exception):
14     def __init__(self, teacher_id):
15         super().__init__(f"Teacher {teacher_id} not found in the system")
16
17 class PaymentValidationException(Exception):
18     def __init__(self, message="Payment validation failed"):
19         super().__init__(message)
20
21 class InvalidStudentDataException(Exception):
22     def __init__(self, message="Invalid student data"):
23         super().__init__(message)
24
25 class InvalidCourseDataException(Exception):
26     def __init__(self, message="Invalid course data"):
27         super().__init__(message)
28
29 class InvalidEnrollmentDataException(Exception):
30     def __init__(self, message="Invalid enrollment data"):
31         super().__init__(message)
32
33 class InvalidTeacherDataException(Exception):
34     def __init__(self, message="Invalid teacher data"):
35         super().__init__(message)
36
37 class InsufficientFundsException(Exception):
38     def __init__(self, student_id, course_id):
39         super().__init__(f"Student {student_id} does not have sufficient funds to enroll in Course {course_id}")
40
```

Driver Program:

```
driver.py x
driver.py > @main
1 from datetime import datetime
2 from student import Student
3 from course import Course
4 from teacher import Teacher
5 from enrollment import Enrollment
6 from payment import Payment
7 from sis import SIS
8
9 def main():
10     try:
11         # Create instances of SIS, Student, Course, and Teacher
12         sis = SIS()
13         student = Student(1, 'rohan', 'Doe', datetime(2000, 1, 1), 'john.doe@example.com', '123-456-7890')
14         course = Course(101, 'Computer Science', 'CS101', None)
15         teacher = Teacher(1, 'Professor', 'Smith', 'mrs.smith@example.com')
16
17         # Test methods related to the Student class
18         student.enroll_in_course(course.course_id) # Method belongs to the Student class
19         student.update_student_info('John', 'Doe', datetime(2000, 1, 1), 'john.doe@example.com', '807-654-3210') # Method belongs to the Student class
20         student.make_payment(100) # Method belongs to the Student class
21         student.display_student_info() # Method belongs to the Student class
22         enrollments = student.get_enrolled_courses() # Method belongs to the Student class
23         payments = student.get_payment_history() # Method belongs to the Student class
24
25         # Test methods related to the Course class
26         course.assign_teacher(teacher.teacher_id) # Method belongs to the Course class
27         course.update_course_info('Introduction to Computer Science', 4) # Method belongs to the Course class
28         course.display_course_info() # Method belongs to the Course class
29         course_enrollments = course.get_enrollments() # Method belongs to the Course class
30         assigned_teacher = course.get_teacher() # Method belongs to the Course class
31
```

```
25 # Test methods related to the Course class
26 course.assign_teacher(teacher.teacher_id) # Method belongs to the Course class
27 course.update_course_info('Introduction to Computer Science', 4) # Method belongs to the Course class
28 course.display_course_info() # Method belongs to the Course class
29 course_enrollments = course.get_enrollments() # Method belongs to the Course class
30 assigned_teacher = course.get_teacher() # Method belongs to the Course class
31
32 # Test methods related to the SIS class
33 sis.enroll_student_in_course(student.student_id, course.course_id) # Method belongs to the SIS class
34 sis.assign_teacher_to_course(teacher.teacher_id, course.course_id) # Method belongs to the SIS class
35 sis.record_payment(student.student_id, 150) # Method belongs to the SIS class
36 enrollment_report = sis.generate_enrollment_report(course.course_id) # Method belongs to the SIS class
37 payment_report = sis.generate_payment_report(student.student_id) # Method belongs to the SIS class
38 course_statistics = sis.calculate_course_statistics(course.course_id) # Method belongs to the SIS class
39
```



```
driver.py X
driver.py main
40     # Display results
41     print("\nTest Results:")
42     student.display_student_info()
43     course.display_course_info()
44     teacher.display_teacher_info()
45     print("\nEnrollments:")
46     for enrollment in enrollments:
47         print(enrollment)
48     print("\nPayments:")
49     for payment in payments:
50         print(payment)
51     print("\ncourse_enrollments:")
52     for course_enrollment in course_enrollments:
53         print(course_enrollment)
54     print("\nAssigned Teacher:")
55     print(assigned_teacher)
56     print("\nenrollment_report:")
57     print(enrollment_report)
58     print("\npayment_report:")
59     print(payment_report)
60     print("\ncourse_statistics:")
61     print(course_statistics)
62
63 except Exception as e:
64     print(f"Error: {str(e)}")
65
66 finally:
67     # Close database connection
68     sys.conn.close()
69
70 if __name__ == "__main__":
71     main()
72
```

Outputs:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
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
PS D:\Hexaware\Assignment\Assignment 2\Python>
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
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:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\Vsacare\Assignment\Assignment 2\Python & "C:\Program Files\Python112\python.exe" "d:\Vsacare\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, 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)), (23, 1, Decimal('150.00'), datetime.date(2023, 12, 24))]

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

Task(7,8,9,10):

```
task-8-10.py X
task-8-10.py > initialize_database
1 import mysql.connector
2 from mysql.connector import Error
3
4 def initialize_database():
5     try:
6         # Establish a connection to the MySQL server
7         connection = mysql.connector.connect(
8             host='localhost',
9             user='root',
10            password='root'
11        )
12
13        cursor = connection.cursor()
14
15        # Create the SIS database
16        cursor.execute("CREATE DATABASE IF NOT EXISTS sisdb")
17        cursor.execute("USE sisdb")
18
19        # Create Students table
20        cursor.execute("""
21            CREATE TABLE IF NOT EXISTS Students (
22                student_id INT PRIMARY KEY,
23                first_name VARCHAR(255),
24                last_name VARCHAR(255),
25                date_of_birth DATE,
26                email VARCHAR(255),
27                phone_number VARCHAR(15)
28            )
29        """)
30
```

```
task-8-10.py X
task-8-10.py > initialize_database
31
32 # Create Courses table
33 cursor.execute("""
34     CREATE TABLE IF NOT EXISTS Courses (
35         course_id INT PRIMARY KEY,
36         course_name VARCHAR(255),
37         credits INT,
38         teacher_id INT,
39         FOREIGN KEY (teacher_id) REFERENCES Teacher(teacher_id)
40     )
41 """)
42
43 # Create Enrollments table
44 cursor.execute("""
45     CREATE TABLE IF NOT EXISTS Enrollments (
46         enrollment_id INT PRIMARY KEY,
47         student_id INT,
48         course_id INT,
49         enrollment_date DATE,
50         FOREIGN KEY (student_id) REFERENCES Students(student_id),
51         FOREIGN KEY (course_id) REFERENCES Courses(course_id)
52     )
53 """)
54
55 # Create Teacher table
56 cursor.execute("""
57     CREATE TABLE IF NOT EXISTS Teacher (
58         teacher_id INT PRIMARY KEY,
59         first_name VARCHAR(255),
60         last_name VARCHAR(255),
61         email VARCHAR(255)
62     )
63 """)
64
```

```
task-8-10.py X
task-8-10.py > initialize_database
64 # Create Payments table
65 cursor.execute("""
66     CREATE TABLE IF NOT EXISTS Payments (
67         payment_id INT PRIMARY KEY,
68         student_id INT,
69         amount DECIMAL(10, 2),
70         payment_date DATE,
71         FOREIGN KEY (student_id) REFERENCES Students(student_id)
72     )
73 """)
74
75 # Commit changes and close cursor
76 connection.commit()
77 cursor.close()
78
79 except Error as e:
80     print(f"Error: {e}")
81
82 finally:
83     # Close database connection
84     if connection.is_connected():
85         connection.close()
86
87 def retrieve_data(table_name):
88     try:
89         connection = mysql.connector.connect(
90             host='localhost',
91             user='root',
92             password='root',
93             database='sisdh'
94         )
95
96         cursor = connection.cursor()
```

```
task-8-10.py X
task-8-10.py > initialize_database
96 # Example: Retrieve all data from a specific table
97 cursor.execute(f"SELECT * FROM {table_name}")
98 data = cursor.fetchall()
99
100 return data
101
102 except Error as e:
103     print(f"Error: {e}")
104
105 finally:
106     if connection.is_connected():
107         cursor.close()
108         connection.close()
109
110 def insert_data(query, values):
111     try:
112         connection = mysql.connector.connect(
113             host='localhost',
114             user='root',
115             password='root',
116             database='sisdh'
117         )
118
119         cursor = connection.cursor()
120
121         # Example: Insert data into a table
122         cursor.execute(query, values)
123         connection.commit()
124
125 except Error as e:
126     print(f"Error: {e}")
127
128 finally:
129     if connection.is_connected():
130         cursor.close()
131         connection.close()
132
133
```

```
task-8-10.py X
task-8-10.py > initialize_database
135 def update_data(query, values):
136     try:
137         connection = mysql.connector.connect(
138             host='localhost',
139             user='root',
140             password='root',
141             database='sisdb'
142         )
143
144         cursor = connection.cursor()
145
146         # Example: Update data in a table
147         cursor.execute(query, values)
148         connection.commit()
149
150     except Error as e:
151         print(f"Error: {e}")
152
153     finally:
154         if connection.is_connected():
155             cursor.close()
156             connection.close()
157
158 def manage_transaction(queries, values):
159     try:
160         connection = mysql.connector.connect(
161             host='localhost',
162             user='root',
163             password='root',
164             database='sisdb'
165         )
166
167         cursor = connection.cursor()
168
169         # Example: Transaction management
170         connection.start_transaction()
```

```
task-8-10.py X
task-8-10.py > initialize_database
171
172     for i in range(len(queries)):
173         cursor.execute(queries[i], values[i])
174
175     connection.commit()
176
177 except Error as e:
178     print(f"Error: {e}")
179     connection.rollback()
180
181 finally:
182     if connection.is_connected():
183         cursor.close()
184         connection.close()
185
186 def dynamic_query_builder(table_name, columns, conditions, sorting):
187     try:
188         connection = mysql.connector.connect(
189             host='localhost',
190             user='root',
191             password='root',
192             database='sisdb'
193         )
194
195         cursor = connection.cursor()
196
197         # Example: Dynamic query builder
198         query = f"SELECT {' , '.join(columns)} FROM {table_name}"
199
200         if conditions:
201             query += f" WHERE {' AND '.join(conditions)}"
202
203         if sorting:
204             query += f" ORDER BY {' , '.join(sorting)}"
205
206         cursor.execute(query)
207         data = cursor.fetchall()
```


Database Changes:

The screenshot shows a SQL editor window titled "SQL File 2" with the following code:

```
1 • use sisdb;
2 • show tables;
3 • select * from students;
4
```

The results pane below displays the data from the 'students' table:

student_id	first_name	last_name	date_of_birth	email	phone_number
1	John	Doe	2000-11-01	jfn.doe@example.com	987-654-3210
2	Steve	Smith	1992-05-20	steve.smith@email.com	867-543-2109
3	Michael	Johnson	1991-08-10	michael.johnson@gmail.com	555-123-4567
4	Daniel	Wilson	1989-11-02	daniel.brown@email.com	999-888-7777
5	Olivia	Miller	1995-04-12	olivia.miller@gmail.com	333-444-5555
6	Ethan	Davis	1994-09-18	ethan.davis@email.com	666-777-8888
7	Ava	Jones	1993-06-30	ava.jones@gmail.com	444-555-6666
8	Logan	Park	1990-12-05	logan.anderson@email.com	222-333-4444
9	Sophia	Moore	1993-02-28	sophia.moore@gmail.com	888-999-0000
10	John	Doe	1999-08-15	jfn.doe@example.com	123-456-7890

The screenshot shows a SQL editor window titled "SQL File 2" with the following code:

```
1 • use sisdb;
2 • show tables;
3 • select * from teacher;
4
```

The results pane below displays the data from the 'teacher' table:

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

SQL File 2

Limit to 1000 rows

```

1 • use sisdb;
2 • show tables;
3 • select * from courses;
4

```

Result Grid

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

SQL Address

My Snippets

Result Grid

Filter Rows

SQL Editor

Find

Find Type

SQL File 2

Limit to 1000 rows

```

1 • use sisdb;
2 • show tables;
3 • select * from enrollments;
4

```

Result Grid

enrollment_id	student_id	course_id	enrollment_date
1	1	101	2023-04-02
2	2	102	2023-05-03
3	3	103	2023-05-03
4	4	104	2023-10-05
5	5	105	2023-09-06
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	2023-02-10
11	1	107	2023-08-01
12	8	104	2023-05-12
13	7	109	2023-12-10

SQL Address

My Snippets

Result Grid

Filter Rows

SQL Editor

Find

Find Type

SQL File 2

Limit to 1000 rows

```

1 • use sisdb;
2 • show tables;
3 • select * from payments;
4

```

Result Grid

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	100.00	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	100.00	2023-12-24

SQL Address

My Snippets

Result Grid

Filter Rows

SQL Editor

Find

Find Type