STUDENT INFORMATION SYSTEM PYTHON

ANUSH KUMAR

🎯 Objective of the Student Information System Project

This project's goal is to use Python and MySQL to create a solid, interactive, and modular Student Information System (SIS) that makes managing student, course, enrollment, instructor, and payment data easier. The system enables administrators to:

- Add, edit, and retrieve student, instructor, and course data with efficiency.
- Manage the course enrollment process for students.
- Keep track of and document students' payment histories.
- Oversee the teacher's course assignments.
- Create reports, including course statistics, payment reports, and enrollment summaries.
- Offer a menu-driven, user-friendly interface that allows access to all features while ensuring appropriate data validation and exception handling.

The system is appropriate for educational institutions and training facilities because it places an emphasis on modularity through DAO (Data Access Object) classes, data integrity, and a scalable database design.

Python Concepts Used in the SIS Project

1.OOP, or object-oriented programming

To represent entities such as StudentDAO, TeacherDAO, CourseDAO, etc., classes and methods are used.

- Logic is encapsulated in exception classes and DAOs.
- improved code reusability and modularity.

2. Managing Exceptions

Try-except blocks are used to gracefully manage failures.
Particular exclusions such as:

- Database ErrorStudent
- NotFoundException
- The TeacherNotFoundException
- CourseNotFound Error
- Exception to Payment

3. MySQL database connectivity With mysql.connector, you can:

- Establish a connection to the MySQL database.
- Run the INSERT, SELECT, UPDATE, and DELETE SQL queries.

- Manage transactions (rollback, commit).

- The connect_db() function is used to abstract connections.

4. Code for modular programming is separated into several files, or modules:

- Data access logic package (dao).
- exceptions/error handling package.
- db_util.py to establish a database connection.
- encourages maintainability, testing, and clarity in programming.

5. Collections (Dictionaries & Lists)
using dictionaries and lists to handle query results.

Task 1: Define Classes

DESCRIPTION:

In order to construct a Student Information System (SIS), this project applies the ideas of Object-Oriented Programming (OOP). Students, classes, enrollments, instructors, and fees are all managed via the system. It specifies important classes with the proper properties and connections, including `Student`, `Course`, `Enrollment`, `Teacher`, and `Payment`. With the use of organized class-based models and real-world relationships, the SIS makes it possible to track student enrollment in courses, instructor assignments, and payment records.

entity/Course.py

```
class Course:
    def __init__(self, CourseID, CourseName, CourseCode,
AssignedTeacher=None):
        self.__CourseID = CourseID
        self.__CourseName = CourseName
        self.__CourseCode = CourseCode
        self.__AssignedTeacher = AssignedTeacher
        self.__Enrollments = []

    def get_CourseID(self):
        return self.__CourseID
```

```
    def get_CourseName(self):
        return self.__CourseName

    def get_CourseCode(self):
        return self.__CourseCode

    def get_AssignedTeacher(self):
        return self.__AssignedTeacher

    def get_Enrollments(self):
        return [enrollment.Student for enrollment in self.__Enrollments]
```

entity/Enrollment.py

```
class Enrollment:
    enrollment_list = []

    def __init__(self, EnrollmentID, Student, Course, EnrollmentDate):
        self.__EnrollmentID = EnrollmentID
        self.__Student = Student
        self.__Course = Course
```

```
        self.__EnrollmentDate = EnrollmentDate

    def get_EnrollmentID(self):
        return self.__EnrollmentID

    def get_Student(self):
        return self.__Student

    def get_Course(self):
        return self.__Course

    def get_EnrollmentDate(self):
        return self.__EnrollmentDate
```

entity/Payment.py

```
class Payment:
    payment_list = []

    def __init__(self, PaymentID, Student, Amount, PaymentDate):
        self.__PaymentID = PaymentID
        self.__Student = Student
```

```
        self.__Amount = Amount
        self.__PaymentDate = PaymentDate

    def get_PaymentID(self):
        return self.__PaymentID

    def get_Student(self):
        return self.__Student

    def get_Amount(self):
        return self.__Amount

    def get_PaymentDate(self):
        return self.__PaymentDate
```

entity/Student.py

```
from datetime import datetime
from Enrollment import Enrollment
from Payment import Payment

class Student:
```

```
    def __init__(self, StudentID, FirstName, LastName, DateOfBirth,
Email, PhoneNumber):
        self.__StudentID = StudentID
        self.__FirstName = FirstName
        self.__LastName = LastName
        self.__DateOfBirth = DateOfBirth
        self.__Email = Email
        self.__PhoneNumber = PhoneNumber
        self.__Enrollments = []
        self.__Payments = []

    def get_StudentID(self):
        return self.__StudentID

    def get_FirstName(self):
        return self.__FirstName

    def get_LastName(self):
        return self.__LastName

    def get_DateOfBirth(self):
        return self.__DateOfBirth
```

```
    def get_Email(self):
        return self.__Email

    def get_PhoneNumber(self):
        return self.__PhoneNumber

    def get_Enrollments(self):
        return self.__Enrollments

 def get_Payments(self):
        return self.__Payments
```

entity/Teacher.py

```
class Teacher:
    def __init__(self, TeacherID, Name, Email, Expertise):
        self.__TeacherID = TeacherID
        self.__Name = Name
        self.__Email = Email
        self.__Expertise = Expertise
        self.__AssignedCourses = []
```

```
def get_TeacherID(self):
    return self.__TeacherID

def get_Name(self):
    return self.__Name

def get_Email(self):
    return self.__Email

def get_Expertise(self):
    return self.__Expertise

def get_AssignedCourses(self):
    return self.__AssignedCourses
```

Task 2: Implement Constructors

DESCRIPTION:

Every class in the Student Information System (SIS) has a constructor defined to initialize its attributes during object creation. Parameterized constructors are used to establish starting values in classes like Student, Course, Enrollment, Teacher, and Payment. Additionally, the SIS class

now has a default constructor to set up system-level settings and collections.

Task 3: Implement Methods

DESCRIPTION

In order to do important tasks including assigning instructors, creating reports, updating data, and enrolling students in classes, a variety of techniques are used in each class of the Student Information System (SIS). These techniques enable functions including presenting student information, monitoring enrollments, handling payments, and computing course statistics by improving interaction and enabling effective data handling inside the system.

dao/CourseDAO.py

```
from util.db_util import connect_db

from exceptions.DatabaseException import DatabaseException

from exceptions.CourseNotFoundException import
CourseNotFoundException


class CourseDAO:

    def assign_teacher(course_id, teacher_id):

        try:

            conn = connect_db()

            cursor = conn.cursor()
```

```
            cursor.execute("update Courses set TeacherID = %s where
CourseID = %s", (teacher_id, course_id))
            conn.commit()
            if cursor.rowcount == 0:
                raise CourseNotFoundException("Course not found")
        except Exception as e:
            raise DatabaseException(f"Error assigning teacher to course:
{str(e)}")
        finally:
            conn.close()



    def update_course_info(course_id, course_code, course_name):
        try:
            conn = connect_db()
            cursor = conn.cursor()
            cursor.execute("update Courses set CourseCode = %s,
CourseName = %s where CourseID = %s", (course_code, course_name,
course_id))
            conn.commit()
            if cursor.rowcount == 0:
                raise CourseNotFoundException("Course not found")
        except Exception as e:
```

```
            raise DatabaseException(f"Error updating course info: {str(e)}")
        finally:
            conn.close()



    def get_course(course_id):
        try:
            conn = connect_db()
            cursor = conn.cursor()
            cursor.execute("select CourseID, CourseName, CourseCode
from Courses where CourseID = %s", (course_id,))
            course = cursor.fetchone()
            if not course:
                raise CourseNotFoundException("Course not found")
            return course
        except Exception as e:
            raise DatabaseException(f"Error fetching course: {str(e)}")
        finally:
            conn.close()



    def get_teacher(course_id):
```

```
        try:
            conn = connect_db()
            cursor = conn.cursor()
            cursor.execute("select TeacherID from Courses where CourseID
= %s", (course_id,))
            teacher = cursor.fetchone()
            if not teacher or teacher[0] is None:
                raise CourseNotFoundException("No teacher assigned to this
course")
            return teacher[0]
        except Exception as e:
            raise DatabaseException(f"Error fetching teacher for course:
{str(e)}")
        finally:
            conn.close()



    def get_enrollments(course_id):
        try:
            conn = connect_db()
            cursor = conn.cursor()
```

```
            cursor.execute("select s.StudentID, s.FirstName, s.LastName
from Enrollments e join Students s ON e.StudentID = s.StudentID where
e.CourseID = %s ", (course_id,))
            return cursor.fetchall()
        except Exception as e:
            raise DatabaseException(f"Error fetching enrollments: {str(e)}")
        finally:
            conn.close()


    def calculate_course_statistics(course_id):
        try:
            conn = connect_db()
            cursor = conn.cursor()


            cursor.execute("select count(*) from enrollments where courseid
= %s", (course_id,))
            num_enrollments = cursor.fetchone()[0]


            cursor.execute("""
                select coalesce(sum(p.amount), 0)
                from payments p
                join enrollments e on p.studentid = e.studentid
                where e.courseid = %s
```

```
            """, (course_id,))
            total_payments = cursor.fetchone()[0]

            return {
                "Total Enrollments": num_enrollments,
                "Total Payments Received": total_payments
            }
        except Exception as e:
            raise DatabaseException(f"Error calculating course statistics: {str(e)}")
        finally:
            conn.close()
```

dao/EnrollmentDAO.py

```
from util.db_util import connect_db
from exceptions.DatabaseException import DatabaseException
from exceptions.EnrollmentException import EnrollmentException

class EnrollmentDAO:
    def get_student(enrollment_id):
        try:
```

```
            conn = connect_db()
            cursor = conn.cursor()
            cursor.execute("""
            select s.StudentID, s.FirstName, s.LastName
            from Enrollments e
            join Students s on e.StudentID = s.StudentID
            where e.EnrollmentID = %s
            """, (enrollment_id,))
            student = cursor.fetchone()
            if not student:
                raise EnrollmentException("Student associated with this
enrollment not found")
            return student
        except Exception as e:
            raise DatabaseException(f"Error fetching student: {str(e)}")
        finally:
            conn.close()


    def get_course(enrollment_id):
        try:
            conn = connect_db()
            cursor = conn.cursor()
```

```
            cursor.execute("""
            select c.CourseID, c.CourseName
            from Enrollments e
            join Courses c ON e.CourseID = c.CourseID
            where e.EnrollmentID = %s
            """, (enrollment_id,))
            course = cursor.fetchone()
            if not course:
                raise EnrollmentException("Course associated with this
enrollment not found")
            return course
        except Exception as e:
            raise DatabaseException(f"Error fetching course: {str(e)}")
        finally:
            conn.close()



    def enroll_student(student_id, course_id):
        try:
            conn = connect_db()
            cursor = conn.cursor()
            cursor.execute("""
```

```
        insert into Enrollments (StudentID, CourseID, EnrollmentDate)
        values (%s, %s, NOW())
        """, (student_id, course_id))
        conn.commit()
        return cursor.lastrowid
    except Exception as e:
        raise DatabaseException(f"Error enrolling student: {str(e)}")
    finally:
        conn.close()

def generate_enrollment_report(course_id):
    try:
        conn = connect_db()
        cursor = conn.cursor()
        cursor.execute("""
            select S.StudentID, S.FirstName, S.LastName, S.Email
            from Enrollments E
            join Students S on E.StudentID = S.StudentID
            where E.CourseID = %s
          """, (course_id,))
        students = cursor.fetchall()
```

```
            if not students:
                return f"No students enrolled in Course {course_id}."

            report = [f"Enrollment Report for Course {course_id}:"]
            for student in students:
                report.append(f"ID: {student[0]}, Name: {student[1]} {student[2]}, Email: {student[3]}")
            return "\n".join(report)
        except Exception as e:
            raise DatabaseException(f"Error fetching enrollment report: {str(e)}")
        finally:
            conn.close()
```

dao/PaymentDAO.py

```
from util.db_util import connect_db
from dao.StudentDAO import StudentDAO
from exceptions.DatabaseException import DatabaseException
from exceptions.StudentNotFoundException import StudentNotFoundException
from exceptions.PaymentException import PaymentException
```

```
class PaymentDAO:
    def get_student(payment_id):
        try:
            conn = connect_db()
            cursor = conn.cursor()
            cursor.execute("""
            select s.StudentID, s.FirstName, s.LastName
            from Payments p
            join Students s on p.StudentID = s.StudentID
            where p.PaymentID = %s
            """, (payment_id,))
            student = cursor.fetchone()
            if not student:
                raise StudentNotFoundException("Student associated with this payment not found")
            return student
        except Exception as e:
            raise DatabaseException(f"Error fetching student: {str(e)}")
        finally:
            conn.close()
```

```
    def get_payment_amount(payment_id):
        try:
            conn = connect_db()
            cursor = conn.cursor()
            cursor.execute("select Amount from Payments where PaymentID
= %s", (payment_id,))
            amount = cursor.fetchone()
            if not amount:
                raise PaymentException("Payment amount not found")
            return amount[0]
        except Exception as e:
            raise DatabaseException(f"Error fetching payment amount:
{str(e)}")
        finally:
            conn.close()



    def get_payment_date(payment_id):
        try:
            conn = connect_db()
            cursor = conn.cursor()
            cursor.execute("select PaymentDate FROM Payments where
PaymentID = %s", (payment_id,))
```

```
            date = cursor.fetchone()
            if not date:
                raise PaymentException("Payment date not found")
            return date[0]
        except Exception as e:
            raise DatabaseException(f"Error fetching payment date:
{str(e)}")
        finally:
            conn.close()



    def make_payment(student_id, amount):
        try:
            conn = connect_db()
            cursor = conn.cursor()
            cursor.execute("""
                    insert into Payments (StudentID, Amount, PaymentDate)
                    values (%s, %s, NOW())
                """, (student_id, amount))
            conn.commit()
        except Exception as e:
            raise PaymentException(f"Error processing payment: {str(e)}")
```

```
        finally:
            conn.close()



    def get_student_from_payment(payment_id):
        try:
            conn = connect_db()
            cursor = conn.cursor()
            cursor.execute("select StudentID from Payments where
PaymentID = %s", (payment_id,))
            student = cursor.fetchone()
            if not student:
                raise PaymentException("Payment not found")
            return student[0]
        except Exception as e:
            raise DatabaseException(f"Error fetching student from payment:
{str(e)}")
        finally:
            conn.close()


    def generate_payment_report(student_id):
        try:
            payments = StudentDAO.get_payment_history(student_id)
```

```
        if not payments:
            return f"No payments found for Student {student_id}."

        report = [f"Payment Report for Student {student_id}:"]
        for payment in payments:
            report.append(
                f"Payment ID: {payment['PaymentID']}, Amount: {payment['Amount']}, Date: {payment['PaymentDate']}")
        return "\n".join(report)
    except Exception as e:
        raise DatabaseException(f"Error fetching payment report: {str(e)}")
```

dao/StudentDAO.py

```
from util.db_util import connect_db
from exceptions.DatabaseException import DatabaseException
from exceptions.StudentNotFoundException import StudentNotFoundException

class StudentDAO:
    def add_student(first_name, last_name, dob, email, phone):
        try:
```

```
            conn = connect_db()
            cursor = conn.cursor()
            sql = """insert into Students (FirstName, LastName,
DateOfBirth, Email, PhoneNumber)
                        values (%s, %s, %s, %s, %s)"""
            cursor.execute(sql, (first_name, last_name, dob, email, phone))
            conn.commit()
            return cursor.lastrowid
        except Exception as e:
            raise DatabaseException(f"Error adding student: {str(e)}")
        finally:
            conn.close()

    def enroll_in_course(student_id, course_id):
        try:
            conn = connect_db()
            cursor = conn.cursor()
            cursor.execute("""
                insert into Enrollments (StudentID, CourseID,
EnrollmentDate)
                values (%s, %s, NOW())
            """, (student_id, course_id))
            conn.commit()
```

```
        except Exception as e:
            raise DatabaseException(f"Error enrolling student: {str(e)}")
        finally:
            conn.close()



    def update_student_info(student_id, first_name, last_name,
date_of_birth, email, phone):
        try:
            conn = connect_db()
            cursor = conn.cursor()
            cursor.execute("""
                update Students
                set FirstName = %s, LastName = %s, DateOfBirth = %s,
Email = %s, PhoneNumber = %s
                where StudentID = %s
            """, (first_name, last_name, date_of_birth, email, phone,
student_id))
            conn.commit()
            if cursor.rowcount == 0:
                raise StudentNotFoundException("Student not found")
        except Exception as e:
            raise DatabaseException(f"Error updating student info: {str(e)}")
```

```
        finally:
            conn.close()


    def get_enrolled_courses(student_id):
        try:
            conn = connect_db()
            cursor = conn.cursor()
            cursor.execute("""
                select c.CourseID, c.CourseName from Enrollments e
                join Courses c ON e.CourseID = c.CourseID
                where e.StudentID = %s
            """, (student_id,))
            return cursor.fetchall()
        except Exception as e:
            raise DatabaseException(f"Error fetching enrolled courses:
{str(e)}")
        finally:
            conn.close()


    def get_student(student_id):
```

```
        try:
            conn = connect_db()
            cursor = conn.cursor()
            cursor.execute(
                "select StudentID, FirstName, LastName, Email,
PhoneNumber from Students where StudentID = %s",
                (student_id,))
            student = cursor.fetchone()
            if not student:
                raise StudentNotFoundException("Student not found")
            return {
                "StudentID": student[0],
                "FirstName": student[1],
                "LastName": student[2],
                "Email": student[3],
                "PhoneNumber": student[4]
            }
        except Exception as e:
            raise DatabaseException(f"Error fetching student: {str(e)}")
        finally:
            conn.close()
```

```
    def get_payment_history(student_id):
        try:
            conn = connect_db()
            cursor = conn.cursor()
            cursor.execute("select PaymentID, Amount, PaymentDate from
Payments where StudentID = %s", (student_id,))
            payments = cursor.fetchall()
            return [{
                "PaymentID": payment[0],
                "Amount": payment[1],
                "PaymentDate": payment[2]
            } for payment in payments]
        except Exception as e:
            raise DatabaseException(f"Error fetching payment history:
{str(e)}")
        finally:
            conn.close()
```

dao/TeacherDAO.py

```
from util.db_util import connect_db
from exceptions.DatabaseException import DatabaseException
```

```
from exceptions.TeacherNotFoundException import
TeacherNotFoundException


class TeacherDAO:


    def add_teacher(name, email, expertise):
        try:
            conn = connect_db()
            cursor = conn.cursor()
            sql = """insert into Teachers (Name, Email, Expertise)
                        values (%s, %s, %s)"""
            cursor.execute(sql, (name, email, expertise))
            conn.commit()
            return cursor.lastrowid
        except Exception as e:
            raise DatabaseException(f"Error adding teacher: {str(e)}")
        finally:
            conn.close()



    def update_teacher_info(teacher_id, name, email, expertise):
        try:
```

```
        conn = connect_db()
        cursor = conn.cursor()
        cursor.execute("""
            update Teachers
            set Name = %s, Email = %s, Expertise = %s
            where TeacherID = %s
        """, (name, email, expertise, teacher_id))
        conn.commit()
        if cursor.rowcount == 0:
            raise TeacherNotFoundException("Teacher not found")
    except Exception as e:
        raise DatabaseException(f"Error updating teacher info: {str(e)}")
    finally:
        conn.close()


def get_assigned_courses(teacher_id):
    try:
        conn = connect_db()
        cursor = conn.cursor()
        cursor.execute("""
```

```
            select CourseID, CourseName from Courses where TeacherID
= %s
            """, (teacher_id,))
            return cursor.fetchall()
        except Exception as e:
            raise DatabaseException(f"Error fetching assigned courses:
{str(e)}")
        finally:
            conn.close()



    def get_teacher(teacher_id):
        try:
            conn = connect_db()
            cursor = conn.cursor()
            cursor.execute("select TeacherID, Name, Email, Expertise from
Teachers where TeacherID = %s", (teacher_id,))
            teacher = cursor.fetchone()
            if not teacher:
                raise TeacherNotFoundException("Teacher not found")
            return {
                "TeacherID": teacher[0],
                "Name": teacher[1],
```

```
            "Email": teacher[2],
            "Expertise": teacher[3]
        }
    except Exception as e:
        raise DatabaseException(f"Error fetching teacher: {str(e)}")
    finally:
        conn.close()
```

Task 4: Exceptions handling and Custom Exceptions

DESCRIPTION:

In order to address certain issue scenarios in the Student Information System (SIS), this activity entails building and utilizing custom exceptions. To enforce business rules and offer relevant error messages, custom exception classes such as `DuplicateEnrollmentException`, `CourseNotFoundException`, and `PaymentValidationException` are developed. By throwing these exceptions in response to actions such as duplicate enrollments, missing data, or erroneous inputs, the system's error handling is improved in terms of control, clarity, and resilience.

exceptions/CourseNotfoundException.py

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

```
        super().__init__(self.message)
```

exceptions/DatabaseException.py

```
class DatabaseException(Exception):
    def __init__(self, message="Database operation failed"):
        self.message = message
        super().__init__(self.message)
```

exceptions/EnrollmentException.py

```
class EnrollmentException(Exception):
    def __init__(self, message="Enrollment operation failed"):
        self.message = message
        super().__init__(self.message)
```

exceptions/PaymentException.py

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

exceptions/StudentNotFoundException.py

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

exceptions/TeacherNotFoundException.py

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

Task 5: Collections

DESCRIPTION:

The goal of this work is to use collections, such as lists, to create links between classes in the Student Information System (SIS). To manage associations like student enrollments, course enrollments, teacher-assigned courses, and student payment records, class-level collections are

used. Constructors are updated to initialize these collections upon object creation, and these relationships are essential for precise data tracking.

Task 6: Create Methods for Managing Relationships

DESCRIPTION:

In order to complete this objective, the SIS class must implement methods for managing relationships and processes, including student enrollment, instructor course assignment, payment recording, and data retrieval. Proper linkage between objects is ensured by methods such as AddEnrollment, AssignCourseToTeacher, and AddPayment. Additionally, a driver program is developed as a console application to create instances, call functions, and handle errors using the Main method in order to test and illustrate the system's capabilities.

Main/main.py

```
from util.db_util import connect_db

from exceptions.DatabaseException import DatabaseException

from exceptions.CourseNotFoundException import
CourseNotFoundException

from exceptions.TeacherNotFoundException import
TeacherNotFoundException

from exceptions.PaymentException import PaymentException
```

```
from exceptions.StudentNotFoundException import
StudentNotFoundException
from dao.StudentDAO import StudentDAO
from dao.TeacherDAO import TeacherDAO
from dao.CourseDAO import CourseDAO
from dao.EnrollmentDAO import EnrollmentDAO
from dao.PaymentDAO import PaymentDAO


def main():
    while True:
        print("\nStudent Information System:")
        print("1. Enroll Student in Course")
        print("2. Update Student Info")
        print("3. Make Payment")
        print("4. Display Student Info")
        print("5. Get Enrolled Courses")
        print("6. Get Payment History")
        print("7. Assign Teacher to Course")
        print("8. Update Course Info")
        print("9. Display Course Info")
        print("10. Get Enrollments")
        print("11. Get Teacher")
```

```
print("12. Get Student from Enrollment")
print("13. Get Course from Enrollment")
print("14. Update Teacher Info")
print("15. Display Teacher Info")
print("16. Get Assigned Courses")
print("17. Get Student from Payment")
print("18. Get Payment Amount")
print("19. Get Payment Date")
print("20. Generate Enrollment Report")
print("21. Generate Payment Report")
print("22. Calculate Course Statistics")
print("23. Exit")

choice = input("Enter: ")

try:
    if choice == "1":
        student_id = input("Enter Student ID: ")
        course_id = input("Enter Course ID: ")
        EnrollmentDAO.enroll_student(student_id, course_id)
        print("Student enrolled successfully.")
    elif choice == "2":
```

```
            student_id = input("Enter Student ID: ")
            first_name = input("Enter First Name: ")
            last_name = input("Enter Last Name: ")
            dob = input("Enter Date of Birth (Y-M-D): ")
            email = input("Enter Email: ")
            phone = input("Enter Phone Number: ")
            StudentDAO.update_student_info(student_id, first_name,
last_name, dob, email, phone)
            print("Student information updated successfully.")
        elif choice == "3":
            student_id = input("Enter Student ID: ")
            amount = float(input("Enter Payment Amount: "))
            PaymentDAO.make_payment(student_id, amount)
            print("Payment recorded successfully.")
        elif choice == "4":
            student_id = input("Enter Student ID: ")
            student_info = StudentDAO.get_student(student_id)
            print(f"ID: {student_info['StudentID']}, Name:
{student_info['FirstName']} {student_info['LastName']}")
            print(f"Email: {student_info['Email']}, Phone:
{student_info['PhoneNumber']}")
        elif choice == "5":
            student_id = input("Enter Student ID: ")
```

```
            courses = StudentDAO.get_enrolled_courses(student_id)
            for course in courses:
                print(f"Course ID: {course[0]}, Name: {course[1]}")
        elif choice == "6":
            student_id = input("Enter Student ID: ")
            payments = StudentDAO.get_payment_history(student_id)
            for payment in payments:
                print(f"Payment ID: {payment['PaymentID']}, Amount:
{payment['Amount']}, Date: {payment['PaymentDate']}")
        elif choice == "7":
            teacher_id = input("Enter Teacher ID: ")
            course_id = input("Enter Course ID: ")
            CourseDAO.assign_teacher(teacher_id, course_id)
            print("Teacher assigned successfully.")
        elif choice == "8":
            course_id = input("Enter Course ID: ")
            course_code = input("Enter Course Code: ")
            course_name = input("Enter Course Name: ")
            CourseDAO.update_course_info(course_id, course_code,
course_name)
            print("Course information updated successfully.")
        elif choice == "9":
            course_id = input("Enter Course ID: ")
```

```
            course_info = CourseDAO.get_course(course_id)
            print(f"Course ID: {course_info[0]}, Name: {course_info[1]},
Code: {course_info[2]}")
        elif choice == "10":
            course_id = input("Enter Course ID: ")
            enrollments = CourseDAO.get_enrollments(course_id)
            for enrollment in enrollments:
                print(f"Enrollment ID: {enrollment[0]}, Student ID:
{enrollment[1]}")
        elif choice == "11":
            course_id = input("Enter Course ID: ")
            teacher = CourseDAO.get_teacher(course_id)
            print(f"Assigned Teacher: {teacher}")
        elif choice == "12":
            enrollment_id = input("Enter Enrollment ID: ")
            student = EnrollmentDAO.get_student(enrollment_id)
            print(f"Student ID: {student}")
        elif choice == "13":
            enrollment_id = input("Enter Enrollment ID: ")
            course = EnrollmentDAO.get_course(enrollment_id)
            print(f"Course ID: {course}")
        elif choice == "14":
            teacher_id = input("Enter Teacher ID: ")
```

```
            name = input("Enter Name: ")
            email = input("Enter Email: ")
            expertise = input("Enter Expertise: ")
            TeacherDAO.update_teacher_info(teacher_id, name, email,
expertise)
            print("Teacher information updated successfully.")
        elif choice == "15":
            teacher_id = input("Enter Teacher ID: ")
            teacher_info = TeacherDAO.get_teacher(teacher_id)
            print(f"Teacher ID: {teacher_info['TeacherID']}, Name:
{teacher_info['Name']}")
        elif choice == "16":
            teacher_id = input("Enter Teacher ID: ")
            courses = TeacherDAO.get_assigned_courses(teacher_id)
            for course in courses:
                print(f"Course ID: {course[0]}, Name: {course[1]}")
        elif choice == "17":
            payment_id = input("Enter Payment ID: ")
            student_id =
PaymentDAO.get_student_from_payment(payment_id)
            print(f"Student ID: {student_id}")
        elif choice == "18":
            payment_id = input("Enter Payment ID: ")
```

```
            amount = PaymentDAO.get_payment_amount(payment_id)
            print(f"Payment Amount: {amount}")
        elif choice == "19":
            payment_id = input("Enter Payment ID: ")
            payment_date =
PaymentDAO.get_payment_date(payment_id)
            print(f"Payment Date: {payment_date}")
        elif choice == "20":
            course_id = input("Enter Course ID: ")
            report =
EnrollmentDAO.generate_enrollment_report(course_id)
            print(report)
        elif choice == "21":
            student_id = input("Enter Student ID: ")
            report = PaymentDAO.generate_payment_report(student_id)
            print(report)
        elif choice == "22":
            course_id = input("Enter Course ID: ")
            stats = CourseDAO.calculate_course_statistics(course_id)
            print(f"Total Enrollments: {stats['Total Enrollments']}")
            print(f"Total Payments Received: {stats['Total Payments
Received']}")
        elif choice == "23":
```

```
            print("Exiting Student Information System.")
            break
        else:
            print("Invalid choice. Please try again.")
    except (DatabaseException, StudentNotFoundException,
TeacherNotFoundException, CourseNotFoundException,
PaymentException) as e:
        print(f"Error: {str(e)}")


main()
```

OUTPUT:

```
Student Information System:
1. Add Student
2. Enroll Student in Course
3. Update Student Info
4. Make Payment
5. Display Student Info
6. Get Enrolled Courses
7. Get Payment History
8. Assign Teacher to Course
9. Update Course Info
10. Display Course Info
11. Get Enrollments
12. Get Teacher
13. Get Student from Enrollment
14. Get Course from Enrollment
15. Update Teacher Info
16. Display Teacher Info
17. Get Assigned Courses
18. Get Student from Payment
19. Get Payment Amount
20. Get Payment Date
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter:
```

Task 7: Database Connectivity

DESCRIPTION:

The integration of database connection into the Student Information System (SIS) is the main objective of this job. It entails adding the necessary tables to the database at startup, putting procedures in place for data entry, retrieval, and updates, and guaranteeing data integrity through error-handling and validation. The purpose of transaction management is to ensure uniformity in processes such as payments and enrollments. In order to enable secure and adaptable SQL queries depending on user input, including conditions, filters, and sorting, a dynamic query builder is also built.

Database Connection:

util/db_util.py

```
import mysql.connector


def connect_db():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="root@123",
        database="SISDB1"
    )
```

Task 8: Student Enrollment:

DESCRIPTION:

Enrolling John Doe, a new student, in the Student Information System (SIS) is the task at hand. John's personal information is entered into the system and kept in the database. Additionally, it generates enrollment records for Mathematics 101 and Introduction to Programming. By adding student and enrollment information utilizing accepted connection and data handling techniques, the job illustrates database interaction.

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**

OUTPUT:

```
6. Get Enrolled Courses
7. Get Payment History
8. Assign Teacher to Course
9. Update Course Info
10. Display Course Info
11. Get Enrollments
12. Get Teacher
13. Get Student from Enrollment
14. Get Course from Enrollment
15. Update Teacher Info
16. Display Teacher Info
17. Get Assigned Courses
18. Get Student from Payment
19. Get Payment Amount
20. Get Payment Date
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 1
Enter First Name: John
Enter Last Name: Doe
Enter Date of Birth (YYYY-MM-DD): 1995-08-15
Enter Email: john.doe@example.com
Enter Phone Number: 1234567890
Student added successfully with ID: 1
```

John is enrolling in the following courses:

- Course 1: Introduction to Programming
- Course 2: Mathematics 101

Task 9: Teacher Assignment

Description:

Assigning Sarah Smith, a new instructor, to the Advanced Database Management (CS302) course in the Student Information System (SIS) is the work at hand. Using the course code, the system obtains the course record, generates a teacher record with her information, and modifies the database course to reflect the designated instructor. This job illustrates how to use database procedures to link instructor and course data.

The system should perform the following tasks:

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

OUTPUT:

```
8. Assign Teacher to Course
9. Update Course Info
10. Display Course Info
11. Get Enrollments
12. Get Teacher
13. Get Student from Enrollment
14. Get Course from Enrollment
15. Update Teacher Info
16. Display Teacher Info
17. Get Assigned Courses
18. Get Student from Payment
19. Get Payment Amount
20. Get Payment Date
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 10
Enter Course ID: 1
Course ID: 1, Name: Introduction to Programming, Code: CS101
```

• Assign Sarah Smith as the instructor for the course.

OUTPUT:

```
9. Update Course Info
10. Display Course Info
11. Get Enrollments
12. Get Teacher
13. Get Student from Enrollment
14. Get Course from Enrollment
15. Update Teacher Info
16. Display Teacher Info
17. Get Assigned Courses
18. Get Student from Payment
19. Get Payment Amount
20. Get Payment Date
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 8
Enter Teacher ID: 1
Enter Course ID: 1
Teacher assigned successfully.
```

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

OUTPUT:

```
10. Display Course Info
11. Get Enrollments
12. Get Teacher
13. Get Student from Enrollment
14. Get Course from Enrollment
15. Update Teacher Info
16. Display Teacher Info
17. Get Assigned Courses
18. Get Student from Payment
19. Get Payment Amount
20. Get Payment Date
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 9
Enter Course ID: 1
Enter Course Code: cs101
Enter Course Name: mathematic 101
Course information updated successfully.
```

Task 10: Payment Record

DESCRIPTION:

This assignment entails entering Jane Johnson's payment into the Student Information System (SIS). Using Jane's student ID, the system fetches her record, enters the payment information (amount and date) into the database, and adjusts her outstanding balance appropriately. This procedure guarantees that student financial transactions are accurately tracked within the system.

• Retrieve John Doe's student record from the database based on her student ID.

OUTPUT:

```
17. Get Assigned Courses
18. Get Student from Payment
19. Get Payment Amount
20. Get Payment Date
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 5
Enter Student ID: 1
ID: 1, Name: John Doe
Email: john.doe@example.com, Phone: 1234567890
```

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

OUTPUT:

```
14. Get Course from Enrollment
15. Update Teacher Info
16. Display Teacher Info
17. Get Assigned Courses
18. Get Student from Payment
19. Get Payment Amount
20. Get Payment Date
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 4
Enter Student ID: 1
Enter Payment Amount: 50000
Payment recorded successfully.
```

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

OUTPUT:

```
14. Get Course from Enrollment
15. Update Teacher Info
16. Display Teacher Info
17. Get Assigned Courses
18. Get Student from Payment
19. Get Payment Amount
20. Get Payment Date
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 7
Enter Student ID: 1
Payment ID: 1, Amount: 50000.00, Date: 2025-04-05
```

Task 11: Enrollment Report Generation

DESCRIPTION:

In the Student Information System (SIS), this operation entails creating an enrollment report for the subject Advance Database 101. The system creates a report with a list of the registered students after retrieving all enrollment information for the designated course from the database. After that, the report is either kept for administrative use or presented.

Course to generate the report for:

• Course Name: Advance Database 101

OUTPUT:

```
10. Display Course Info
11. Get Enrollments
12. Get Teacher
13. Get Student from Enrollment
14. Get Course from Enrollment
15. Update Teacher Info
16. Display Teacher Info
17. Get Assigned Courses
18. Get Student from Payment
19. Get Payment Amount
20. Get Payment Date
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 23
Enter Course ID: 2
Total Enrollments: 0
Total Payments Received: 0.00
```

• Retrieve enrollment records from the database for the specified course.

OUTPUT:

```
12. Get Teacher
13. Get Student from Enrollment
14. Get Course from Enrollment
15. Update Teacher Info
16. Display Teacher Info
17. Get Assigned Courses
18. Get Student from Payment
19. Get Payment Amount
20. Get Payment Date
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 21
Enter Course ID: 1
No students enrolled in Course 1.
```

Update Student Information:

```
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 3
Enter Student ID: 1
Enter First Name: Anush
Enter Last Name: Kumar
Enter Date of Birth (Y-M-D): 2003-10-19
Enter Email: anush@gmail.com
Enter Phone Number: 123654789
Student information updated successfully.
```

Get Payment History:

```
17. Get Assigned Courses
18. Get Student from Payment
19. Get Payment Amount
20. Get Payment Date
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 7
Enter Student ID: 1
Payment ID: 1, Amount: 50000.00, Date: 2025-04-05
```

Display Course:

```
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 10
Enter Course ID: 1
Course ID: 1, Name: mathematic 101, Code: cs101
```

Get Assigned Teacher:

```
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 12
Enter Course ID: 1
Assigned Teacher: 1
```

Get Payment Date:

```
19. Get Payment Amount
20. Get Payment Date
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 20
Enter Payment ID: 1
Payment Date: 2025-04-05
```

Get Payment Amount:

```
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 19
Enter Payment ID: 1
Payment Amount: 50000.00
```

Get Student from Payment:

```
20. Get Payment Date
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 18
Enter Payment ID: 1
Student ID: 1
```

Get Assigned Course:

```
19. Get Payment Amount
20. Get Payment Date
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 17
Enter Teacher ID: 1
Course ID: 1, Name: mathematic 101
```

Display Teacher Information:

```
17. Get Assigned Courses
18. Get Student from Payment
19. Get Payment Amount
20. Get Payment Date
21. Generate Enrollment Report
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 16
Enter Teacher ID: 1
Teacher ID: 1, Name: Sarah Smith
```

Update Teacher Information:

```
22. Generate Payment Report
23. Calculate Course Statistics
24. Exit
Enter: 15
Enter Teacher ID: 1
Enter Name: lana
Enter Email: lana@gmail
Enter Expertise: AI
Teacher information updated successfully.
```