STUDENT INFORMATION SYSTEM

ASSIGNMENT

dao:

1.courses.py

```
from util.dbConnection import dbConnection
from exception.exception import *
from dao.teachers import Teachers


# COURSES CLASS
class Courses(dbConnection):
    # CONSTRUCTOR
    def __init__(self):
        self.courseID = int()
        self.courseName = str()
        self.credits = int()
        self.teacherID = int()

    # CRUD METHODS
    # METHOD TO PERFORM SELECT OPERATION
    def select(self):
        self.open()
        select_str = '''select * from Courses'''
        self.stmt = self.conn.cursor()
        self.stmt.execute(select_str)
        data = self.stmt.fetchall()
        return data

    # THIS METHOD WILL CREATE THE TABLE
    def create(self):
        self.open()
        create_str = '''CREATE TABLE if not exists Courses (
                        courseID INT PRIMARY KEY,
                        courseName VARCHAR(50),
                        credits INT,
                        teacherID INT,
                        FOREIGN KEY (teacherID) REFERENCES Teachers(teacherID) on
delete cascade)'''
        self.stmt.execute(create_str)
        self.close()

    # THIS METHOD WILL INSERT DATA INTO THE TABLE
    def insert(self, courseID, courseName, credits, teacherID):
        self.courseID = courseID
        self.courseName = courseName
        self.courseCode = credits
        self.teacherID = teacherID
        self.open()
        insert_str = f'''INSERT INTO Courses VALUES
           ({courseID}, '{courseName}', '{credits}', '{teacherID}')'''
        try:
            self.stmt.execute(insert_str)
            self.conn.commit()
        except Exception as e:
            print(str(e) + "---INCORRECT TYPE OF DATA---")
        self.close()

    # THIS METHOD WILL UPDATE THE DATA
    def update(self, courseID, courseName, credits, teacherID):
```

```
        self.courseID = courseID
        self.courseName = courseName
        self.credits = credits
        self.teacherID = teacherID
        self.open()
        data = [(courseName, credits, teacherID, courseID)]
        update_str = '''UPDATE courses
           set courseName = %s, credits = %s, teacherID = %s
           where courseID = %s'''
        try:
            self.stmt.executemany(update_str, data)
            self.conn.commit()
        except Exception as e:
            print(str(e) + "---INCORRECT TYPE OF DATA---")
        self.close()

    # THIS METHOD WILL DELETE THE RECORD
    def delete(self, courseID):
        self.courseID = courseID
        delete_str = f'''delete from courses where courseID = {courseID}'''
        self.open()
        self.stmt.execute(delete_str)
        self.conn.commit()
        self.close()

    # THIS METHOD WILL ASSIGN TEACHER TO A COURSE
    def AssignTeacher(self, teacher: Teachers, teacherID, courseID):
        try:
            data = teacher.select()
            for i in data:
                if i[0] == teacherID:
                    break
            else:
                raise InvalidTeacherDataException
        except InvalidTeacherDataException as e:
            print(e)
        else:
            try:
                data = self.select()
                for i in data:
                    if i[0] == courseID:
                        break
                else:
                    raise InvalidCourseDataException
            except InvalidCourseDataException as e:
                print(e)
            else:
                data = self.select()
                for i in data:
                    if i[0] == courseID and i[3] == teacherID:
                        print("TEACHER ALREADY ASSIGNED")
                        break
                else:
                    query = f'''UPDATE COURSES SET TEACHERID = {teacherID} WHERE
COURSEID = {courseID}'''
                    self.open()
                    self.stmt = self.conn.cursor()
                    self.stmt.execute(query)
                    self.conn.commit()
                    self.close()
                    print("TEACHER ASSIGNED")

    # THIS METHOD WILL UPDATE THE COURSE INFORMATION
    def UpdateCourseInfo(self, courseID: int, courseName: str, teacherID: int,
teacher:Teachers):
        try:
            data = teacher.select()
```

```
            for i in data:
                if i[0] == teacherID:
                    break
            else:
                raise InvalidTeacherDataException
        except InvalidTeacherDataException as e:
            print(e)
        else:
            try:
                data = self.select()
                for i in data:
                    if i[0] == courseID:
                        break
                else:
                    raise InvalidCourseDataException
            except InvalidCourseDataException as e:
                print(e)
            else:
                query = f'''UPDATE COURSES SET COURSENAME = %s, TEACHERID = %s WHERE
COURSEID = %s'''
                data = [(courseName,teacherID,courseID)]
                self.open()
                self.stmt = self.conn.cursor()
                self.stmt.executemany(query, data)
                self.conn.commit()
                self.close()
                print("UPDATED SUCCESSFULLY")

    # THIS METHOD WILL DISPLAY PARTICULAR COURSE INFORMATION
    def DisplayCourseInfo(self):
        ID = int(input("Enter Course ID : "))
        data = self.select()
        try:
            for i in data:
                if i[0] == ID:
                    print(i)
                    break
            else:
                raise CourseNotFoundException
        except CourseNotFoundException as e:
            print(e)

    # THIS METHOD WILL GET ALL THE ENROLLMENTS
    def GetEnrollments(self):
        ID = int(input("Enter Course ID : "))
        try:
            data = self.select()
            for i in data:
                if i[0] == ID:
                    break
            else:
                raise CourseNotFoundException
        except CourseNotFoundException as e:
            print(e)
        else:
            query = f'''SELECT  C.coursename, S.studentid, S.firstname, S.lastname,
E.ENROLLMENT_DATE
                        FROM Enrollments AS E
                        INNER JOIN Students AS S ON E.studentid = S.studentid
                        INNER JOIN Courses AS C ON C.courseid = E.courseid
                        WHERE C.COURSEID = {ID};'''
            self.open()
            self.stmt = self.conn.cursor()
            self.stmt.execute(query)
            data = self.stmt.fetchall()
            self.close()
            for i in data:
```

```
                print(i)

    # THIS METHOD WILL GET DETAILS OF A PARTICULAR TEACHER
    def GetTeacher(self):
        ID = int(input("Enter Course ID : "))
        try:
            data = self.select()
            for i in data:
                if i[0] == ID:
                    break
            else:
                raise CourseNotFoundException
        except CourseNotFoundException as e:
            print(e)
        else:
            query = f'''SELECT C.COURSENAME, T.FIRSTNAME, T.LASTNAME FROM COURSES AS C
            INNER JOIN TEACHERS AS T ON C.TEACHERID = T.TEACHERID
            WHERE COURSEID = {ID} GROUP BY C.COURSENAME'''
            self.open()
            self.stmt = self.conn.cursor()
            self.stmt.execute(query)
            data = self.stmt.fetchall()
            self.close()
            for i in data:
                print(i)

c=Courses()
c.create()
c.insert(11,'python','3',1)
c.insert(12,'sql','2',3)
c.insert(13,'java','3',4)
c.insert(14,'azure','3',98)
```

2.enrollments.py

```
from util.dbConnection import dbConnection
from exception.exception import *
from datetime import datetime


# ENROLLMENT CLASS
class Enrollments(dbConnection):
    # CONSTRUCTOR
    def __init__(self):
        self.enrollmentID = int()
        self.studentID = int()
        self.courseID = int()
        self.enrollmentDate = datetime.date

    # CRUD METHODS
    # SELECT METHOD TO GET THE DATA
    def select(self):
        self.open()
        select_str = '''select * from Enrollments'''
        self.stmt = self.conn.cursor()
        self.stmt.execute(select_str)
        data = self.stmt.fetchall()
        return data

    # CREATE METHOD TO CREATE THE TABLE
    def create(self):
        self.open()
        create_str = '''CREATE TABLE if not exists Enrollments (
                        enrollmentID INT PRIMARY KEY,
                        studentID INT, FOREIGN KEY (studentID) REFERENCES
```

```
Students(studentID) on delete cascade,
                        courseID INT, FOREIGN KEY (courseID) REFERENCES
Courses(courseID) on delete cascade,
                        enrollment_date DATE);'''
        self.stmt = self.conn.cursor()
        self.stmt.execute(create_str)
        self.close()

    # INSERT FUNCTION TO INSERT THE DATA
    def insert(self, enrollmentID, studentID, courseID, enrollmentDate):
        self.enrollmentID = enrollmentID
        self.studentID = studentID
        self.courseID = courseID
        self.enrollmentDate = enrollmentDate
        self.open()
        insert_str = f'''INSERT INTO Enrollments VALUES
        ({enrollmentID}, {studentID}, {courseID}, '{enrollmentDate}')'''
        self.stmt = self.conn.cursor()
        try:
            self.stmt.execute(insert_str)
            self.conn.commit()
        except Exception as e:
            print(str(e) + "---INCORRECT TYPE OF DATA---")
        self.close()

    # UPDATE FUNCTION TO UPDATE THE DATA
    def update(self, enrollmentID, studentID, courseID, enrollmentDate):
        self.enrollmentID = enrollmentID
        self.studentID = studentID
        self.courseID = courseID
        self.enrollmentDate = enrollmentDate
        self.open()
        data = [(studentID, courseID, enrollmentDate, enrollmentID)]
        update_str = '''UPDATE enrollments
        set studentID = %s, courseID = %s, enrollmentDate = %s
        where enrollmentID = %s'''
        self.stmt = self.conn.cursor()
        try:
            self.stmt.executemany(update_str, data)
            self.conn.commit()
        except Exception as e:
            print(str(e) + "---INCORRECT TYPE OF DATA---")
        self.close()

    # DELETE FUNCTION TO DELETE A RECORD
    def delete(self, enrollmentID):
        self.enrollmentID = enrollmentID
        delete_str = f'''delete from enrollments where enrollmentID = {enrollmentID}'''
        self.open()
        self.stmt = self.conn().cursor()
        self.stmt.execute(delete_str)
        self.conn.commit()
        self.close()

    # FUNCTION TO GET STUDENT DETAILS
    def GetStudent(self):
        ID = int(input("Enter Enrollment ID : "))
        try:
            data = self.select()
            for i in data:
                if i[0] == ID:
                    break
            else:
                raise InvalidEnrollmentDataException
        except InvalidEnrollmentDataException as e:
            print(e)
        else:
```

```
            query = f'''SELECT S.FIRSTNAME, S.LASTNAME FROM STUDENTS AS S
            INNER JOIN ENROLLMENTS AS E ON S.STUDENTID = E.STUDENTID
            WHERE E.ENROLLMENTID = {ID}'''
            self.open()
            self.stmt = self.conn.cursor()
            self.stmt.execute(query)
            data = self.stmt.fetchall()
            self.close()
            print(data)

    # FUNCTION TO GET COURSE DETAILS
    def GetCourse(self):
        ID = int(input("Enter Enrollment ID : "))
        try:
            data = self.select()
            for i in data:
                if i[0] == ID:
                    break
            else:
                raise InvalidEnrollmentDataException
        except InvalidEnrollmentDataException as e:
            print(e)
        else:
            query = f'''SELECT C.COURSENAME FROM COURSES AS C
                    INNER JOIN ENROLLMENTS AS E ON C.COURSEID = E.COURSEID
                    WHERE E.ENROLLMENTID = {ID}'''
            self.open()
            self.stmt = self.conn.cursor()
            self.stmt.execute(query)
            data = self.stmt.fetchall()
            self.close()
            print(data)


e=Enrollments()
e.create()
e.insert(101,1,11,'2023-09-01')
e.insert(102,2,12,'2023-09-04')
e.insert(103,3,13,'2023-09-07')
e.insert(104,4,14,'2023-10-01')
```

3.payments.py

```
import datetime

from util.dbConnection import dbConnection


# PAYMENTS CLASS
class Payments(dbConnection):
    # CONSTRUCTOR
    def __init__(self):
        self.paymentID = int()
        self.studentID = int()
        self.amount = int()
        self.paymentDate = datetime.date

    # CRUD METHODS
    # SELECT METHOD TO GET ALL THE DATA
    def select(self):
        self.open()
        select_str = '''select * from Payments'''
        self.stmt = self.conn.cursor()
        self.stmt.execute(select_str)
```

```
        data = self.stmt.fetchall()
        return data

    # CREATE METHOD TO CREATE THE TABLE
    def create(self):
        self.open()
        create_str = '''CREATE TABLE if not exists Payments (
                        paymentID INT PRIMARY KEY,
                        studentID INT, FOREIGN KEY (studentID) REFERENCES
Students(studentID) on delete cascade,
                        amount INT,
                        paymentDate DATE);'''
        self.stmt = self.conn.cursor()
        self.stmt.execute(create_str)
        self.close()

    # INSERT METHOD TO INSERT THE DATA
    def insert(self, paymentID, studentID, amount, paymentDate):
        self.paymentID = paymentID
        self.studentID = studentID
        self.amount = amount
        self.paymentDate = paymentDate
        self.open()
        insert_str = f'''INSERT INTO Payments VALUES
        ({paymentID}, {studentID}, {amount}, '{paymentDate}')'''
        self.stmt = self.conn.cursor()
        try:
            self.stmt.execute(insert_str)
            self.conn.commit()
        except Exception as e:
            print(str(e) + "---INCORRECT TYPE OF DATA---")
        self.close()

    # UPDATE METHOD TO UPDATE THE RECORDS
    def update(self, paymentID, studentID, amount, paymentDate):
        self.paymentID = paymentID
        self.studentID = studentID
        self.amount = amount
        self.paymentDate = paymentDate
        self.open()
        data = [(studentID, amount, paymentDate, paymentID)]
        update_str = '''UPDATE payments
        set studentID = %s, amount = %s, paymentDate = %s
        where paymentID = %s'''
        self.stmt = self.conn.cursor()
        try:
            self.stmt.executemany(update_str, data)
            self.conn.commit()
        except Exception as e:
            print(str(e) + "---INCORRECT TYPE OF DATA---")
        self.close()

    # DELETE METHOD TO DELETE THE RECORDS
    def delete(self, paymentID):
        self.paymentID = paymentID
        delete_str = f'''delete from payments where paymentID = {paymentID}'''
        self.open()
        self.stmt = self.conn().cursor()
        self.stmt.execute(delete_str)
        self.conn.commit()
        self.close()

    # FUNCTION TO GET STUDENT DETAILS
    def GetStudent(self, ID):
        try:
            data = self.select()
            for i in data:
```

```
                if i[1] == ID:
                    break
            else:
                raise Exception
        except Exception as e:
            print("INVALID DATA")
        else:
            query = f'''SELECT S.FIRSTNAME, S.LASTNAME, P.PAYMENTID, P.PAYMENTDATE
                    FROM STUDENTS AS S
                    INNER JOIN PAYMENTS AS P ON S.STUDENTID = P.STUDENTID
                    WHERE S.STUDENTID = {ID};'''
            self.open()
            self.stmt = self.conn.cursor()
            self.stmt.execute(query)
            data = self.stmt.fetchall()
            self.close()
            print(data)

    # FUNCTION TO GET PAYMENT AMOUNT
    def GetPaymentAmount(self):
        ID = int(input("Enter Payment ID : "))
        try:
            data = self.select()
            for i in data:
                if i[0] == ID:
                    break
            else:
                raise Exception
        except Exception as e:
            print("INVALID PAYMENT ID")
        else:
            query = f'''SELECT AMOUNT FROM PAYMENTS
                    WHERE PAYMENTID = {ID}'''
            self.open()
            self.stmt = self.conn.cursor()
            self.stmt.execute(query)
            data = self.stmt.fetchall()
            self.close()
            print(data)

    # FUNCTION TO GET PAYMENT DATE
    def GetPaymentDate(self):
        ID = int(input("Enter Payment ID : "))
        try:
            data = self.select()
            for i in data:
                if i[0] == ID:
                    break
            else:
                raise Exception
        except Exception as e:
            print("INVALID PAYMENT ID")
        else:
            query = f'''SELECT paymentDate FROM PAYMENTS
                            WHERE PAYMENTID = {ID}'''
            self.open()
            self.stmt = self.conn.cursor()
            self.stmt.execute(query)
            data = self.stmt.fetchall()
            self.close()
            print(data)

p=Payments()
p.create()
p.insert(1,1,23456,'2023-10-10')
p.insert(2,2,25000,'2023-02-05')
```

```
p.insert(3,3,10000,'2023-09-30')
p.insert(4,4,24000,'2023-11-01')
```

4.students.py

```
from util.dbConnection import dbConnection
from datetime import datetime
from exception.exception import *


# STUDENTS CLASS
class Students(dbConnection):
    # CONSTRUCTOR
    def __init__(self):
        self.studentID = int()
        self.firstName = str()
        self.lastName = str()
        self.dateOfBirth = datetime.date
        self.email = str()
        self.phoneNumber = str()

    # CRUD METHODS
    # SELECT FUNCTION TO GET ALL THE DATA
    def select(self) -> list:
        self.open()
        select_str = '''select * from Students'''
        self.stmt = self.conn.cursor()
        self.stmt.execute(select_str)
        data = self.stmt.fetchall()
        return data

    # CREATE FUNCTION TO CREATE THE TABLE
    def create(self):
        self.open()
        create_str = '''CREATE TABLE if not exists Students (studentID INT PRIMARY KEY,
        firstName VARCHAR(50),
        lastName VARCHAR(50),
        dateOfBirth DATE,
        email VARCHAR(50),
        phoneNumber CHAR(10));'''
        self.stmt = self.conn.cursor()
        self.stmt.execute(create_str)
        self.close()

    # INSERT FUNCTION TO INSERT THE DATA
    def insert(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.open()
        insert_str = f'''INSERT INTO Students VALUES
        ({studentID}, '{firstName}', '{lastName}', '{dateOfBirth}', '{email}',
'{phoneNumber}')'''
        self.stmt = self.conn.cursor()
        try:
            self.stmt.execute(insert_str)
            self.conn.commit()
        except Exception as e:
            print("---DATA ALREADY EXISTS---")
        self.close()

    # UPDATE FUNCTION TO UPDATE THE DATA
    def update(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.open()
        data = [(firstName, lastName, dateOfBirth, email, phoneNumber, studentID)]
        update_str = '''UPDATE students
        set firstName = %s, lastName = %s, dateOfBirth = %s, email = %s, phoneNumber =
%s
        where studentID = %s'''
        self.stmt = self.conn.cursor()
        try:
            self.stmt.executemany(update_str, data)
            self.conn.commit()
            print("Updated Successfully")
        except Exception as e:
            print(str(e) + "---INCORRECT TYPE OF DATA---")
        self.close()

    # DELETE FUNCTION TO DELETE THE RECORDS
    def delete(self, studentID):
        self.studentID = studentID
        delete_str = f'''delete from students where studentID = {studentID}'''
        self.open()
        self.stmt = self.conn().cursor()
        self.stmt.execute(delete_str)
        self.conn.commit()
        self.close()

    # FUNCTION TO ENROLL STUDENT IN A COURSE
    def EnrollInCourse(self, courseID: int):
        self.courseID = courseID
        self.studentID = 7
        self.open()
        self.stmt = self.conn.cursor()
        select_str = f'select * from enrollments where studentID = {self.studentID} and
courseID = {courseID}'
        self.stmt.execute(select_str)
        data = self.stmt.fetchall()
        if len(data) == 1:
            try:
                raise DuplicateEnrollmentException
            except DuplicateEnrollmentException as e:
                print(e)
        else:
            try:
                id = int(input("Enter a UNIQUE enrollment ID : "))
                enroll_course = f'insert into enrollments values({id},
{self.studentID}, {courseID}, CURDATE())'
                self.stmt.execute(enroll_course)
            except Exception as e:
                print('---ENROLLMENT ID EXISTS/ ALREADY ENROLLED---')
            self.conn.commit()
            print("Enrolled successfully")
        self.close()

    # FUNCTION TO UPDATE THE STUDENT DETAILS
    def UpdateStudentInfo(self, firstName: str, lastName: str, dateOfBirth: datetime,
email: str, phoneNumber: str):
        ID = int(input("Enter student ID : "))
        self.update(studentID=
ID,firstName=firstName,lastName=lastName,dateOfBirth=dateOfBirth,
                    email=email, phoneNumber=phoneNumber)

    # FUNCTION TO MAKE PAYMENT
```

```
    def MakePayment(self, amount: int, paymentDate: datetime):
        ID = int(input("Enter student ID : "))
        self.open()
        self.stmt = self.conn.cursor()
        while(True):
            try:
                pID = int(input("Enter a Unique payment ID"))
                payment_str = f'insert into payments values({pID}, {ID}, {amount},' \

f'{paymentDate.year}{paymentDate.month}{paymentDate.day})'
                self.stmt.execute(payment_str)
                self.conn.commit()
                print("Inserted Successfully")
            except Exception as e:
                print(str(e)+'---PAYMENT ID EXISTS---')
            else:
                break
        self.close()

    # FUNCTION TO DISPLAY STUDENT DETAILS
    def DisplayStudentInfo(self):
        data = self.select()
        try:
            ID = int(input("Enter Student ID : "))
            for i in data:
                if i[0] == ID:
                    print(i)
                    break
            else:
                raise StudentNotFoundException
        except StudentNotFoundException as e:
            print(e)

    # FUNCTION TO FETCH ENROLLED COURSES OF A STUDENT
    def GetEnrolledCourses(self):
        self.open()
        self.stmt = self.conn.cursor()
        ID = int(input("Enter Student ID : "))
        try:
            data = self.select()
            for i in data:
                if i[0] == ID:
                    break
            else:
                raise StudentNotFoundException
        except StudentNotFoundException as e:
            print(e)
        else:
            getCourses = f'''SELECT courseName FROM Students AS S
                             INNER JOIN Enrollments AS E ON S.studentID=E.studentID
                             INNER JOIN Courses as C ON C.CourseID=E.CourseID
                             WHERE S.studentID = {ID}
                             GROUP BY S.studentID'''
            self.stmt.execute(getCourses)
            data = self.stmt.fetchall()
            print(data)
        finally:
            self.close()

    # FUNCTION TO FETCH PAYMENT HISTORY OF A STUDENT
    def GetPaymentHistory(self, ID):
        try:
            data = self.select()
            for i in data:
                if i[0] == ID:
                    break
            else:
```

```
                raise StudentNotFoundException
        except StudentNotFoundException as e:
            print(e)
        else:
            self.open()
            self.stmt = self.conn.cursor()
            paymentHistory = f'''SELECT PAYMENTID, AMOUNT, PAYMENTDATE FROM PAYMENTS
                                 WHERE STUDENTID = {ID}'''
            self.stmt.execute(paymentHistory)
            data = self.stmt.fetchall()
            self.close()
            print('PAYMENT_ID, AMOUNT, DATE')
            for i in data:
                print(i)

s=Students()
s.create()
s.insert(1,'neeha','choudary','2001-06-07','neeha@email.com','9099783562')
s.insert(2,'rahul','singh','1999-07-12','rahul@email.com','8097654389')
s.insert(3,'muskan','saxena','2001-09-28','muskan@email.com','7899367273')
s.insert(4,'vaishnavi','doneti','2002-05-09','vaishnavi@email.com','6782398892')
```

5.teachers.py

```
from util.dbConnection import dbConnection
from exception.exception import *


# TEACHERS CLASS
class Teachers(dbConnection):
    def __init__(self):
        self.teacherID = int()
        self.firstName = str()
        self.lastName = str()
        self.email = str()

    # CRUD METHODS
    # SELECT FUNCTION TO GET THE DATA
    def select(self):
        self.open()
        select_str = '''select * from Teachers'''
        self.stmt = self.conn.cursor()
        self.stmt.execute(select_str)
        data = self.stmt.fetchall()
        return data

    # CREATE FUNCTION TO CREATE THE TABLE
    def create(self):
        self.open()
        create_str = '''CREATE TABLE if not exists Teachers (
                        teacherID INT PRIMARY KEY,
                        firstName VARCHAR(50),
                        lastName VARCHAR(50),
                        email VARCHAR(50));'''
        self.stmt = self.conn.cursor()
        self.stmt.execute(create_str)
        self.close()

    # INSERT FUNCTION TO INSERT THE DATA
    def insert(self, teacherID, firstName, lastName, email):
        self.teacherID = teacherID
        self.firstName = firstName
        self.lastName = lastName
        self.email = email
        self.open()
        insert_str = f'''INSERT INTO Teachers VALUES
```

```
        ({teacherID}, '{firstName}', '{lastName}', '{email}')'''
        self.stmt = self.conn.cursor()
        try:
            self.stmt.execute(insert_str)
            self.conn.commit()
        except Exception as e:
            print(str(e) + "---INCORRECT TYPE OF DATA OR DATA EXISTS---")
        self.close()

    # UPDATE FUNCTION TO UPDATE THE DATA
    def update(self, teacherID, firstName, lastName, email):
        self.teacherID = teacherID
        self.firstName = firstName
        self.lastName = lastName
        self.email = email
        self.open()
        data = [(firstName, lastName, email, teacherID)]
        update_str = '''UPDATE Teachers
        set firstName = %s, lastName = %s, email = %s
        where teacherID = %s'''
        self.stmt = self.conn.cursor()
        try:
            self.stmt.executemany(update_str, data)
            self.conn.commit()
        except Exception as e:
            print(str(e) + "---INCORRECT TYPE OF DATA---")
        self.close()

    # DELETE FUNCTION TO DELETE A RECORD
    def delete(self, teacherID):
        self.teacherID = teacherID
        delete_str = f'''delete from teachers where teacherID = {teacherID}'''
        self.open()
        self.stmt = self.conn().cursor()
        self.stmt.execute(delete_str)
        self.conn.commit()
        self.close()

    # FUNCTION TO UPDATE TEACHER DETAILS
    def UpdateTeacherInfo(self, firstName : str, lastName : str, email : str):
        ID = int(input("Enter teacher ID : "))
        self.update(ID, firstName, lastName, email)
        print('UPDATED SUCCESSFULLY')

    # FUNCTION TO DISPLAY TEACHER DETAILS
    def DisplayTeacherInfo(self):
        ID = int(input("Enter Teacher ID : "))
        data = self.select()
        try:
            for i in data:
                if i[0] == ID:
                    print(i)
                    break
            else:
                raise TeacherNotFoundException
        except TeacherNotFoundException as e:
            print(e)

    # FUNCTION TO GET ASSIGNED COURSES OF A TEACHER
    def GetAssignedCourses(self):
        ID = int(input("Enter Teacher ID : "))
        data = self.select()
        try:
            for i in data:
                if i[0] == ID:
                    break
            else:
```

```
                raise TeacherNotFoundException
        except TeacherNotFoundException as e:
            print(e)
        else:
            query = f'''SELECT t.teacherid,t.firstname,t.lastname,c.coursename FROM
Courses AS C
                        INNER JOIN Teachers AS T ON C.teacherid=T.teacherid
                        where t.teacherid = {ID}
                        GROUP BY t.teacherid,t.firstname,t.lastname,c.coursename'''
            self.open()
            self.stmt = self.conn.cursor()
            self.stmt.execute(query)
            data = self.stmt.fetchall()
            self.close()
            for i in data:
                print(i)


t=Teachers()
t.create()
t.insert(98,'Shalini','Pandey','shalini@email.com')
t.insert(2,'Neha','gupta','gupta@email.com')
t.insert(3,'Hari','krishn','krishn@email.com')
t.insert(4,'santosh','shaik','shaik@email.com')
```

6.tasks.py

```
from dao.students import Students
from dao.teachers import Teachers
from dao.courses import Courses
from dao.enrollments import Enrollments
from dao.payments import Payments


# OBJECT CREATION
s = Students()
t = Teachers()
c = Courses()
e = Enrollments()
p = Payments()


# TASK 8 - ADDING A STUDENT JOHN AND ENROLLING HIM IN TWO COURSES
def studentEnrollment():
    s.insert(10, 'Jonny', 'bush', '1995-08-15', 'johndoe@example.com', '1234567890')
    s.EnrollInCourse(103)
    s.EnrollInCourse(104)


# TASK 9 - ADDING A NEW TEACHER AND ASSIGNING HIM/HER TO A COURSE
def teacherAssignment():
    t.insert(8, 'Sarah', 'Smith', 'sarah.smith@example.com')
    c.AssignTeacher(t, 8, 102)


# TASK 10 - RECORDS THE PAYMENT IN THE DATABASE
def paymentRecord():
    p.GetStudent(1)
    s.GetPaymentHistory(1)


# TASK 11 - ENROLLMENT REPORT GENERATION
def enrollmentReportGeneration():
    data = c.GetEnrollments()
```

```
# THIS METHOD WILL FETCH STUDENT DATA
def studentInfo():
    data = s.select()
    for i in data:
        print(i)
    return data


# THIS METHOD WILL FETCH TEACHER DATA
def teacherInfo():
    data = t.select()
    for i in data:
        print(i)
    return data


# THIS METHOD WILL FETCH COURSE DATA
def courseInfo():
    data = c.select()
    for i in data:
        print(i)
    return data


# THIS METHOD WILL FETCH ENROLLMENT DATA
def enrollmentInfo():
    data = e.select()
    for i in data:
        print(i)
    return data


# THIS METHOD WILL FETCH PAYMENT DATA
def paymentInfo():
    data = p.select()
    for i in data:
        print(i)
    return data
```

entity:

1.courses.py

```
class Courses():
    # CONSTRUCTOR
    def __init__(self):
        self.courseID = int()
        self.courseName = str()
        self.credits = int()
        self.teacherID = int()
```

2.enrollments.py

```
import datetime


class Enrollments():
    # CONSTRUCTOR
    def __init__(self):
        self.enrollmentID = int()
        self.studentID = int()
        self.courseID = int()
        self.enrollmentDate = datetime.date()
```

3.payments.py

```
import datetime


class Payments():
    # CONSTRUCTOR
    def __init__(self):
        self.paymentID = int()
        self.studentID = int()
        self.amount = int()
        self.paymentDate = datetime.date()
```

4.students.py

```
from datetime import datetime


class Students():
    # CONSTRUCTOR
    def __init__(self):
        self.studentID = int()
        self.firstName = str()
        self.lastName = str()
        self.dateOfBirth = datetime.date()
        self.email = str()
        self.phoneNumber = str()
```

5.teachers.py

```
class Teachers():
    # CONSTRUCTOR
    def __init__(self):
        self.teacherID = int()
        self.firstName = str()
        self.lastName = str()
        self.email = str()
```

exception:

exception.py

```
# THIS EXCEPTION WILL RAISE WHEN ENROLLMENT ALREADY EXISTS
class DuplicateEnrollmentException(Exception):
    def __init__(self,msg="ENROLLMENT ALREADY EXISTS"):
        super().__init__(msg)


# THIS EXCEPTION WILL RAISE WHEN COURSE ALREADY EXISTS
class CourseNotFoundException(Exception):
    def __init__(self,msg="COURSE NOT FOUND"):
        super().__init__(msg)


# THIS EXCEPTION WILL RAISE WHEN STUDENT DO NOT EXIST
class StudentNotFoundException(Exception):
    def __init__(self,msg="STUDENT NOT FOUND"):
```

```
        super().__init__(msg)


# THIS EXCEPTION WILL RAISE WHEN TEACHER DO NOT EXIST
class TeacherNotFoundException(Exception):
    def __init__(self,msg="TEACHER NOT FOUND"):
        super().__init__(msg)


# THIS EXCEPTION WILL RAISE WHEN PAYMNET IS INVALID
class PaymentValidationException(Exception):
    def __init__(self,msg="INVALID PAYMENT AMOUNT/DATE"):
        super().__init__(msg)


# THIS EXCEPTION WILL RAISE WHEN STUDENT DATA IS INVALID
class InvalidStudentDataException(Exception):
    def __init__(self,msg="INVALID STUDENT DATA"):
        super().__init__(msg)


# THIS EXCEPTION WILL RAISE WHEN COURSE DATA IS INVALID
class InvalidCourseDataException(Exception):
    def __init__(self,msg="INVALID COURSE DATA"):
        super().__init__(msg)


# THIS EXCEPTION WILL RAISE WHEN ENROLLMENT DATA IS INVALID
class InvalidEnrollmentDataException(Exception):
    def __init__(self,msg="INVALID ENROLLMENT DATA"):
        super().__init__(msg)


# THIS EXCEPTION WILL RAISE WHEN TEACHER DATA IS INVALID
class InvalidTeacherDataException(Exception):
    def __init__(self,msg="INVALID TEACHER DATA"):
        super().__init__(msg)


# THIS EXCEPTION WILL RAISE WHEN STUDENT HAS INSUFFICIENT FUNDS
class InsufficientFundsException(Exception):
    def __init__(self,msg="STUDENT HAS INSUFFICIENT FUNDS"):
        super().__init__(msg)
```

util:

dbConnection.py

```
import mysql.connector as sql


class dbConnection:
    # THIS FUNCTION WILL GET THE CONNECTION
    def open(self):
        try:
            self.conn = sql.connect(host='localhost', database='sisdb', user='root',
password='sweetysmiley')
            self.stmt = self.conn.cursor()
        except Exception as e:
            print(str(e) + '-----DATABASE NOT FOUND-----')

    # THIS FUNCTION  WILL CLOSE THE CONNECTION
    def close(self):
        try:
```

```
            self.conn.close()
        except Exception:
            pass
```

main:

main.py

```
from dao.tasks import *


print(" WELCOME TO SIS ")
while(True):
    print("\n select 1 to perform task 8 - ADDING A STUDENT JOHN AND ENROLLING HIM IN
TWO COURSES")
    print(" 2 to perform task 9 - ADDING A NEW TEACHER AND ASSIGNING HIM/HER TO A
COURSE")
    print(" 3 to perform task 10 - RECORDS THE PAYMENT IN THE DATABASE")
    print(" 4 to perform task 11 - ENROLLMENT REPORT GENERATION")
    print(" 5 to get all the teacher data")
    print(" 6 to get all the student data")
    print(" 7 to get all the payments data")
    print(" 8 to get all the courses data")
    print(" 9 to get all the enrollments data")
    print(" 10 to exit")
    s = input("Enter your choice : ")
    if s == '1':
        try:
            studentEnrollment()
        except Exception as e:
            print()
    elif s == '2':
        try:
            teacherAssignment()
        except Exception as e:
            print()
    elif s == '3':
        try:
            paymentRecord()
        except Exception as e:
            print()
    elif s == '4':
        try:
            enrollmentReportGeneration()
        except Exception as e:
            print()
    elif s == '5':
        teacherInfo()
    elif s == '6':
        studentInfo()
    elif s == '7':
        paymentInfo()
    elif s == '8':
        courseInfo()
    elif s == '9':
        enrollmentInfo()
    elif s == '10':
        print("*"*30 + " THANK YOU FOR USING STUDENT INFORMATION SYSTEM " + "*"*30)
        break
    else:
        print("Invalid choice")
```

Outputs:

```
select 1 to perform task 8 - ADDING A STUDENT JOHN AND ENROLLING HIM IN TWO COURSES
2 to perform task 9 - ADDING A NEW TEACHER AND ASSIGNING HIM/HER TO A COURSE
3 to perform task 10 - RECORDS THE PAYMENT IN THE DATABASE
4 to perform task 11 - ENROLLMENT REPORT GENERATION
5 to get all the teacher data
6 to get all the student data
7 to get all the payments data
8 to get all the courses data
9 to get all the enrollments data
10 to exit
Enter your choice :
```

```
Enter your choice : 1
Enter a UNIQUE enrollment ID: 104
Enrolled successfully
```

```
Enter your choice : 2
TEACHER ASSIGNED
```

```
Enter your choice : 3
[('neeha', 'choudary', 1, datetime.date(2023, 10, 10))]
PAYMENT_ID, AMOUNT, DATE
(1, 23456, datetime.date(2023, 10, 10))
```

```
Enter your choice : 4
Enter Course ID : 14
('azure', 4, 'vaishnavi', 'doneti', datetime.date(2023, 10, 1))
```

```
Enter your choice : 5
(1, 'Shalini', 'Pandey', 'shalini@email.com')
(2, 'Neha', 'gupta', 'gupta@email.com')
(3, 'Hari', 'krishn', 'krishn@email.com')
(4, 'santosh', 'shaik', 'shaik@email.com')
(5, 'Sarah', 'Smith', 'sarah.smith@example.com')
(6, 'Sarah', 'Smith', 'sarah.smith@example.com')
(7, 'Sarah', 'Smith', 'sarah.smith@example.com')
(8, 'Sarah', 'Smith', 'sarah.smith@example.com')
(11, 'Saif', 'khan', 'skh@example.com')
(12, 'ali', 'khan', 'skh@example.com')
(13, 'alia', 'bhatt', 'ab@example.com')
(98, 'Shalini', 'Pandey', 'shalini@email.com')
```

```
Enter your choice : 6
(1, 'neeha', 'choudary', datetime.date(2001, 6, 7), 'neeha@email.com', '9099783562')
(2, 'rahul', 'singh', datetime.date(1999, 7, 12), 'rahul@email.com', '8097654389')
(3, 'muskan', 'saxena', datetime.date(2001, 9, 28), 'muskan@email.com', '7899367273')
(4, 'vaishnavi', 'doneti', datetime.date(2002, 5, 9), 'vaishnavi@email.com', '6782398892')
(6, 'John', 'Doe', datetime.date(1995, 8, 15), 'johndoe@example.com', '1234567890')
(7, 'John', 'Doe', datetime.date(1995, 8, 15), 'johndoe@example.com', '1234567890')
(10, 'Jonny', 'bush', datetime.date(1995, 8, 15), 'johndoe@example.com', '1234567890')
(11, 'Joseph', 'bush', datetime.date(1999, 8, 15), 'joe@example.com', '1234567890')
```

```
Enter your choice : 7
(1, 1, 23456, datetime.date(2023, 10, 10))
(2, 2, 25000, datetime.date(2023, 2, 5))
(3, 3, 10000, datetime.date(2023, 9, 30))
(4, 4, 24000, datetime.date(2023, 11, 1))
```

```
Enter your choice : 8
(11, 'python', 3, 1)
(12, 'sql', 2, 3)
(13, 'java', 3, 4)
(14, 'azure', 3, 98)
```

```
Enter your choice : 9
(101, 1, 11, datetime.date(2023, 9, 1))
(102, 2, 12, datetime.date(2023, 9, 4))
(103, 3, 13, datetime.date(2023, 9, 7))
(104, 4, 14, datetime.date(2023, 10, 1))
```