SQL & JAVA OOP Assignment – Student Information System

Name: Tejsinh J. Harale-Bhosale

Github Repository: https://github.com/tejsinh3600/DA_foundation

SQL

TASK - 1

1. **Cr**eate the database named "SISDB"

```
create sisdb;
use sisdb;
show tables;
desc students;
```

```
1 •  create database sisdb;
2 •  use sisdb;
```

2. Define the schema for the Students, Courses, Enrollments, Teacher, and Payments tables based on the provided schema. Write SQL scripts to create the mentioned tables with appropriate data types, constraints, and relationships.

a. Students

b. Courses

c. Enrollments

d. Teacher

e. Payments

```
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    date_of_birth DATE,
    email VARCHAR(100) UNIQUE,
    phone_number VARCHAR(15)
);
```

Result Grid | Filter Rows: | Export: | Wrap Cell

Field	Type	Null	Key	Default	Extra
student_id	int	NO	PRI	NULL	
first_name	varchar(45)	NO		NULL	
last_name	varchar(45)	NO		NULL	
date_of_birth	date	NO		NULL	
email	varchar(45)	NO		NULL	
phone_number	varchar(45)	NO		NULL	

```
CREATE TABLE Teacher (
```

```
    teacher_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE
);
```

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default
teacher_id	int	NO	PRI	NULL
first_name	varchar(50)	YES		NULL
last_name	varchar(50)	YES		NULL
email	varchar(100)	YES	UNI	NULL

```
CREATE TABLE Courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(100),
    credits INT,
    teacher_id INT,
    FOREIGN KEY (teacher_id) REFERENCES Teacher(teacher_id)
);
```

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default
course_id	int	NO	PRI	NULL
course_name	varchar(100)	YES		NULL
credits	int	YES		NULL
teacher_id	int	YES	MUL	NULL

```
CREATE TABLE Enrollments (
    enrollment_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    enrollment_date DATE,
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```

Result Grid | Filter Rows: | Exp

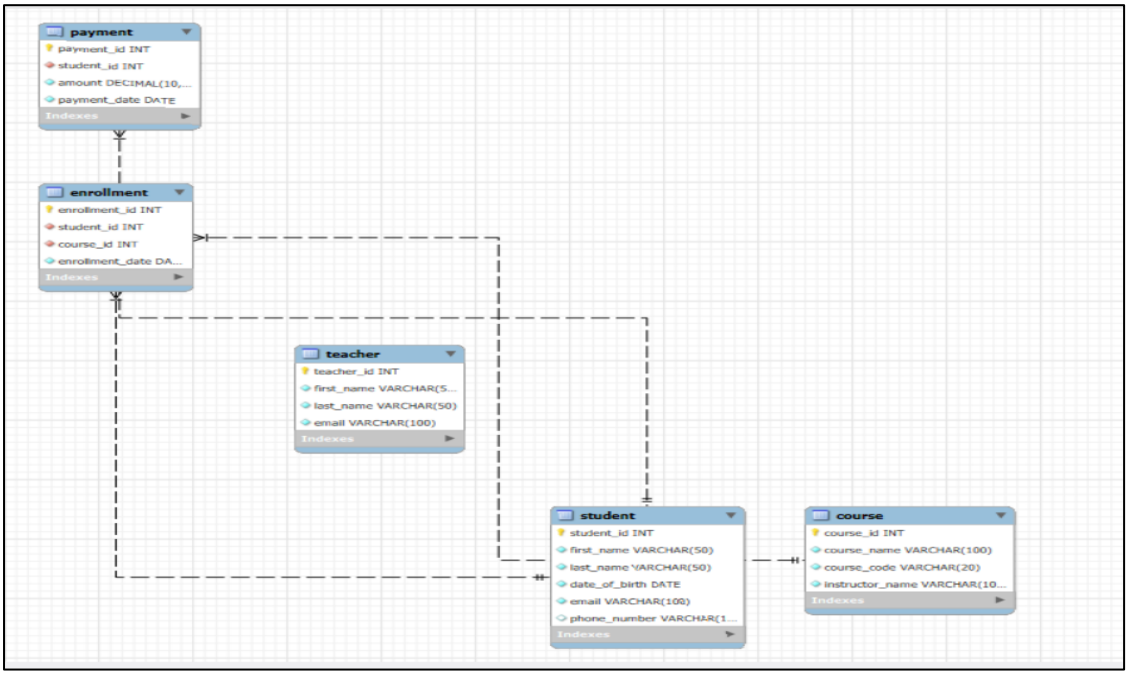
Field	Type	Null	Key	Default
enrollment_id	int	NO	PRI	NULL
student_id	int	YES	MUL	NULL
course_id	int	YES	MUL	NULL
enrollment_date	date	YES		NULL

```
CREATE TABLE Payments (
    payment_id INT PRIMARY KEY,
    student_id INT,
    amount DECIMAL(10, 2),
    payment_date DATE,
    FOREIGN KEY (student_id) REFERENCES Students(student_id)
);
```

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default
payment_id	int	NO	PRI	NULL
student_id	int	YES	MUL	NULL
amount	decimal(10,2)	YES		NULL
payment_date	date	YES		NULL

2. Create an ERD (Entity Relationship Diagram) for the database.



3. Create appropriate Primary Key and Foreign Key constraints for referential integrity.

- As per 1st question

5. Insert at least 10 sample records into each of the following tables.

i. Students

ii. Courses

iii. Enrollments

iv. Teacher

v. Payments

```sql
-- Insert records into Student Table
INSERT INTO Student (first_name, last_name, date_of_birth, email, phone_number) VALUES
('John', 'Doe', '2000-05-15', 'john.doe@example.com', '1234567890'),
('Jane', 'Smith', '1999-08-22', 'jane.smith@example.com', '2345678901'),
('Alice', 'Johnson', '2001-12-05', 'alice.johnson@example.com', '3456789012'),
('Bob', 'Brown', '1998-03-30', 'bob.brown@example.com', '4567890123'),
('Charlie', 'Davis', '2000-01-18', 'charlie.davis@example.com', '5678901234');
```

```sql

-- Teacher
INSERT INTO Teacher (teacher_id, first_name, last_name, email) VALUES
(1, 'Alan', 'Turing', 'alan.turing@example.com'),
(2, 'Grace', 'Hopper', 'grace.hopper@example.com'),
(3, 'Ada', 'Lovelace', 'ada.l@example.com'),
(4, 'Marie', 'Curie', 'marie.curie@example.com'),
(5, 'Nikola', 'Tesla', 'nikola.t@example.com'),
(6, 'Albert', 'Einstein', 'albert.e@example.com'),
(7, 'Katherine', 'Johnson', 'katherine.j@example.com'),
(8, 'Stephen', 'Hawking', 'stephen.h@example.com'),
(9, 'Carl', 'Sagan', 'carl.sagan@example.com'),
(10, 'Jane', 'Goodall', 'jane.g@example.com');
```

```sql
-- Courses
INSERT INTO Courses (course_id, course_name, credits, teacher_id) VALUES
(201, 'Mathematics', 4, 1),
(202, 'Physics', 3, 2),
(203, 'Computer Science', 5, 3),
(204, 'Chemistry', 3, 4),
(205, 'Electrical Engineering', 4, 5),
(206, 'Artificial Intelligence', 5, 6),
(207, 'Astronomy', 3, 7),
(208, 'Quantum Mechanics', 4, 8),
(209, 'Philosophy of Science', 2, 9),
(210, 'Environmental Science', 3, 10);

-- Enrollments
INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date) VALUES
(301, 101, 201, '2024-01-10'),
(302, 102, 202, '2024-01-12'),
(303, 103, 203, '2024-01-15'),
(304, 104, 204, '2024-01-17'),
(305, 105, 205, '2024-01-19'),
(306, 106, 206, '2024-01-21'),
(307, 107, 207, '2024-01-23'),
(308, 108, 208, '2024-01-25'),
(309, 109, 209, '2024-01-27'),
(310, 110, 210, '2024-01-29');
```

```sql
-- Payments
INSERT INTO Payments (payment_id, student_id, amount, payment_date) VALUES
(401, 101, 1500.00, '2024-02-01'),
(402, 102, 1600.00, '2024-02-03'),
(403, 103, 1700.00, '2024-02-05'),
(404, 104, 1800.00, '2024-02-07'),
(405, 105, 1900.00, '2024-02-09'),
(406, 106, 2000.00, '2024-02-11'),
(407, 107, 2100.00, '2024-02-13'),
(408, 108, 2200.00, '2024-02-15'),
(409, 109, 2300.00, '2024-02-17'),
(410, 110, 2400.00, '2024-02-19');
```

Tasks 2: Select, Where, Between, AND, LIKE:

1. Write an SQL query to insert a new student into the "Students" table with the following details:

a. First Name: John

b. Last Name: Doe

c. Date of Birth: 1995-08-15

d. Email: john.doe@example.com

e. Phone Number: 1234567890

```
insert into students (student_id,first_name,last_name,date_of_birth,email,phone_number)
values (101,'John','Doe','1995-08-15','john.doe@example.com','1234567890 ');
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

student_id	first_name	last_name	date_of_birth	email	phone_number
101	John	Doe	1995-08-15	john.doe@example.com	1234567890
102	Emma	Watson	1998-04-15	emma.w@example.com	1234509876

2. Write an SQL query to enroll a student in a course. Choose an existing student and course and insert a record into the "Enrollments" table with the enrollment date.

```
INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date)
VALUES (311, 103, 201, '2026-04-09');
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

enrollment_id	student_id	course_id	enrollment_date
308	108	208	2024-01-25
309	109	209	2024-01-27
310	110	210	2024-01-29
311	103	201	2026-04-09

3. Update the email address of a specific teacher in the "Teacher" table. Choose any teacher and modify their email address.

```
update teacher
set email = 'xyz@abc.com'
where teacher_id = 1;
```

Result Grid | Filter Rows: | Edit

teacher_id	first_name	last_name	email
1	Alan	Turing	xyz@abc.com

4. Write an SQL query to delete a specific enrollment record from the "Enrollments" table. Select an enrollment record based on the student and course.

```
DELETE FROM Enrollments
WHERE student_id = 101 AND course_id = 201;
```

Result Grid | Filter Rows: | Edit:

enrollment_id	student_id	course_id	enrollment_date
302	102	202	2024-01-12

5. Update the "Courses" table to assign a specific teacher to a course. Choose any course and teacher from the respective tables.

```
UPDATE Courses
SET teacher_id = 1
WHERE course_id = 204;
```

Result Grid | Filter Rows: | Edit:

course_id	course_name	credits	teacher_id
201	Mathematics	4	1
202	Physics	3	2
203	Computer Science	5	3
204	Chemistry	3	1

6. Delete a specific student from the "Students" table and remove all their enrollment records from the "Enrollments" table. Be sure to maintain referential integrity.

```
DELETE FROM enrollments
WHERE student_id = 101;
DELETE FROM students
WHERE student_id = 101;
```

Result Grid | Filter Rows: | Edit:

enrollment_id	student_id	course_id	enrollment_date
303	103	203	2024-01-15

7. Update the payment amount for a specific payment record in the "Payments" table. Choose any payment record and modify the payment amount.

```
UPDATE Payments
SET amount = 9999
WHERE payment_id = 402;
```

Result Grid | Filter Rows: | E

payment_id	student_id	amount	payment_date
402	102	9999.00	2024-02-03

Task 3. Aggregate functions, Having, Order By, GroupBy and Joins:

1. Write an SQL query to calculate the total payments made by a specific student. You will need to join the "Payments" table with the "Students" table based on the student's ID.

```sql
select s.first_name, s.last_name, p.amount
from students s
join payments p on s.student_id = p.student_id
where s.student_id = 103;
```

Filter Rows: | Export: | Wrap C

first_name	last_name	amount
Liam	Smith	1700.00

2. Write an SQL query to retrieve a list of courses along with the count of students enrolled in each course. Use a JOIN operation between the "Courses" table and the "Enrollments" table.

```sql
SELECT C.course_name, COUNT(E.student_id) AS student_count
FROM Courses C
LEFT JOIN Enrollments E ON C.course_id = E.course_id
GROUP BY C.course_name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

course_name	student_count
Mathematics	1
Physics	0
Computer Science	1
Chemistry	1
Electrical Engineering	1
Artificial Intelligence	1
Astronomy	1
Quantum Mechanics	1
Philosophy of Science	1
Environmental Science	1

3. Write an SQL query to find the names of students who have not enrolled in any course. Use a LEFT JOIN between the "Students" table and the "Enrollments" table to identify students without enrollments.

```sql
SELECT S.first_name, S.last_name
FROM Students S
LEFT JOIN Enrollments E ON S.student_id = E.student_id
WHERE E.enrollment_id IS NULL;
```

esult Grid | Filter Rows: | Export: | Wrap Cell Conten

first_name	last_name
Emma	Watson

4. Write an SQL query to retrieve the first name, last name of students, and the names of the courses they are enrolled in. Use JOIN operations between the "Students" table and the "Enrollments" and "Courses" tables.

```
SELECT S.first_name, S.last_name, C.course_name
FROM Students S
JOIN Enrollments E ON S.student_id = E.student_id
JOIN Courses C ON E.course_id = C.course_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell

first_name	last_name	course_name
Liam	Smith	Computer Science
Olivia	Johnson	Chemistry
Noah	Brown	Electrical Engineering
Ava	Davis	Artificial Intelligence
William	Wilson	Astronomy
Sophia	Moore	Quantum Mechanics
James	Taylor	Philosophy of Science
Isabella	Anderson	Environmental Science
Liam	Smith	Mathematics

5. Create a query to list the names of teachers and the courses they are assigned to. Join the "Teacher" table with the "Courses" table.

```
SELECT T.first_name, T.last_name, C.course_name
FROM Teacher T
JOIN Courses C ON T.teacher_id = C.teacher_id;
```

Result Grid | Filter Rows: | Export: | Wrap C

first_name	last_name	course_name
Alan	Turing	Mathematics
Alan	Turing	Chemistry
Grace	Hopper	Physics
Ada	Lovelace	Computer Science
Nikola	Tesla	Electrical Engineering
Albert	Einstein	Artificial Intelligence
Katherine	Johnson	Astronomy
Stephen	Hawking	Quantum Mechanics
Carl	Sagan	Philosophy of Science
Jane	Goodall	Environmental Science

6. Retrieve a list of students and their enrollment dates for a specific course. You'll need to join the "Students" table with the "Enrollments" and "Courses" tables.

```
SELECT S.first_name, S.last_name, E.enrollment_date
FROM Students S
JOIN Enrollments E ON S.student_id = E.student_id
JOIN Courses C ON E.course_id = C.course_id
WHERE C.course_id = 201;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Co

first_name	last_name	enrollment_date
Liam	Smith	2026-04-09

7. Find the names of students who have not made any payments. Use a LEFT JOIN between the "Students" table and the "Payments" table and filter for students with NULL payment records.

```
SELECT S.first_name, S.last_name
FROM Students S
LEFT JOIN Payments P ON S.student_id = P.student_id
WHERE P.payment_id IS NULL;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Cor

first_name	last_name
a	p
atharva	patil

8. Write a query to identify courses that have no enrollments. You'll need to use a LEFT JOIN between the "Courses" table and the "Enrollments" table and filter for courses with NULL enrollment records.

```
SELECT C.course_name
FROM Courses C
LEFT JOIN Enrollments E ON C.course_id = E.course_id
WHERE E.enrollment_id IS NULL;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Conte

course_name
Physics

9. Identify students who are enrolled in more than one course. Use a self-join on the "Enrollments" table to find students with multiple enrollment records.

```
SELECT DISTINCT S.student_id, S.first_name, S.last_name
FROM Enrollments E1
JOIN Enrollments E2 ON E1.student_id = E2.student_id AND E1.course_id <> E2.course_id
JOIN Students S ON E1.student_id = S.student_id;
```

id | Filter Rows: | Export: | Wrap Cell Content:

student_id	first_name	last_name
103	Liam	Smith
102	Emma	Watson

10. Find teachers who are not assigned to any courses. Use a LEFT JOIN between the "Teacher" table and the "Courses" table and filter for teachers with NULL course assignments.

```
Select t.first_name, t.last_name
From teacher t
left join courses c on c.teacher_id = t.teacher_id
where c.teacher_id IS NULL;
```

Filter Rows: | Export: | Wrap Cell Co

first_name	last_name
Marie	Curie

Task 4. Subquery and its type:

1. Write an SQL query to calculate the average number of students enrolled in each course. Use aggregate functions and subqueries to achieve this.

```
SELECT AVG(enroll_count) AS avg_students_per_course
FROM (
    SELECT COUNT(student_id) AS enroll_count
    FROM Enrollments
    GROUP BY course_id
) AS course_enrollments;
```

esult Grid | Filter Rows: | Export: | Wrap Cell Co

avg_students_per_course
1.0000

2. Identify the student(s) who made the highest payment. Use a subquery to find the maximum payment amount and then retrieve the student(s) associated with that amount.

```
SELECT S.first_name, S.last_name, P.amount
FROM Payments P
JOIN Students S ON S.student_id = P.student_id
WHERE P.amount = (
    SELECT MAX(amount) FROM Payments
);
```

esult Grid | Filter Rows: | Export: | Wrap

first_name	last_name	amount
Emma	Watson	9999.00

3. Retrieve a list of courses with the highest number of enrollments. Use subqueries to find the course(s) with the maximum enrollment count.

```sql
SELECT C.course_name, COUNT(E.student_id) AS enrollments
FROM Courses C
JOIN Enrollments E ON C.course_id = E.course_id
GROUP BY C.course_id, C.course_name
HAVING COUNT(E.student_id) = (
    SELECT MAX(enroll_count) FROM (
        SELECT COUNT(student_id) AS enroll_count
        FROM Enrollments
        GROUP BY course_id
    ) AS sub
);
```

Filter Rows: | Export: | Wrap Cell Content:

course_name	enrollments
Artificial Intelligence	2

4. Calculate the total payments made to courses taught by each teacher. Use subqueries to sum payments for each teacher's courses.

```sql
SELECT T.first_name, T.last_name, SUM(P.amount) AS total_course_payment
FROM Teacher T
JOIN Courses C ON T.teacher_id = C.teacher_id
JOIN Enrollments E ON C.course_id = E.course_id
JOIN Payments P ON E.student_id = P.student_id
GROUP BY T.teacher_id, T.first_name, T.last_name;
```

Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name	total_course_payment
Alan	Turing	3500.00
Grace	Hopper	9999.00
Ada	Lovelace	1700.00
Nikola	Tesla	1900.00
Albert	Einstein	11999.00
Katherine	Johnson	2100.00
Stephen	Hawking	2200.00
Carl	Sagan	2300.00
Jane	Goodall	0.00

5. Identify students who are enrolled in all available courses. Use subqueries to compare a student's enrollments with the total number of courses.

```sql
SELECT student_id
FROM Enrollments
GROUP BY student_id
HAVING COUNT(DISTINCT course_id) = (SELECT COUNT(*) FROM Courses);
```

Filter Rows: | Export: | Wrap Cell Content:

student_id

6. Retrieve the names of teachers who have not been assigned to any courses. Use subqueries to find teachers with no course assignments.

```
SELECT first_name, last_name
FROM Teacher
WHERE teacher_id NOT IN (
    SELECT DISTINCT teacher_id FROM Courses WHERE teacher_id IS NOT NUL
);
```

Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name
Marie	Curie

7. Calculate the average age of all students. Use subqueries to calculate the age of each student based on their date of birth.

```
SELECT AVG(age) AS avg_age
FROM (
    SELECT TIMESTAMPDIFF(YEAR, date_of_birth, CURDATE()) AS age
    FROM Students
) AS sub;
```

Filter Rows: | Export: | Wrap Cell Content:

avg_age
25.1818

8. Identify courses with no enrollments. Use subqueries to find courses without enrollment records.

```
SELECT course_name
FROM Courses
WHERE course_id NOT IN (
    SELECT DISTINCT course_id FROM Enrollments
);
```

Filter Rows: | Export: | Wrap

course_name
basic tech

9. Calculate the total payments made by each student for each course they are enrolled in. Use subqueries and aggregate functions to sum payments.

```
SELECT E.student_id, E.course_id, SUM(P.amount) AS total_payment
FROM Enrollments E
JOIN Payments P ON E.student_id = P.student_id
GROUP BY E.student_id, E.course_id;
```

Filter Rows: | Export: | Wrap Cell Content:

student_id	course_id	total_payment
102	202	9999.00
102	206	9999.00
103	203	1700.00
103	201	1700.00
104	204	1800.00
105	205	1900.00
106	206	2000.00
107	207	2100.00
108	208	2200.00
109	209	2300.00
110	210	0.00

10. Identify students who have made more than one payment. Use subqueries and aggregate functions to count payments per student and filter for those with counts greater than one.

```
SELECT student_id, COUNT(payment_id) AS num_payments
FROM Payments
GROUP BY student_id
HAVING COUNT(payment_id) > 1;
```

Filter Rows: | Export: | Wrap Cell Conte

student_id	num_payments

11. Write an SQL query to calculate the total payments made by each student. Join the "Students" table with the "Payments" table and use GROUP BY to calculate the sum of payments for each student.

```
SELECT S.first_name, S.last_name, SUM(P.amount) AS total_payment
FROM Students S
JOIN Payments P ON S.student_id = P.student_id
GROUP BY S.student_id, S.first_name, S.last_name;
```

Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name	total_payment
Emma	Watson	9999.00
Liam	Smith	1700.00
Olivia	Johnson	1800.00
Noah	Brown	1900.00
Ava	Davis	2000.00
William	Wilson	2100.00
Sophia	Moore	2200.00
James	Taylor	2300.00
Isabella	Anderson	0.00

12. Retrieve a list of course names along with the count of students enrolled in each course. Use JOIN operations between the "Courses" table and the "Enrollments" table and GROUP BY to count enrollments.

```
SELECT C.course_name, COUNT(E.student_id) AS enrolled_students
FROM Courses C
LEFT JOIN Enrollments E ON C.course_id = E.course_id
GROUP BY C.course_name;
```

Filter Rows: Export: Wrap Cell Content:

course_name	enrolled_students
Mathematics	1
Physics	1
Computer Science	1
Chemistry	1
Electrical Engineering	1
Artificial Intelligence	2
Astronomy	1
Quantum Mechanics	1
Philosophy of Science	1
Environmental Science	1
basic tech	0

13. Calculate the average payment amount made by students. Use JOIN operations between the "Students" table and the "Payments" table and GROUP BY to calculate the average.

```
SELECT S.student_id, S.first_name, S.last_name, AVG(P.amount) AS avg_payment
FROM Students S
JOIN Payments P ON S.student_id = P.student_id
GROUP BY S.student_id, S.first_name, S.last_name;
```

Filter Rows: Export: Wrap Cell Content:

student_id	first_name	last_name	avg_payment
102	Emma	Watson	9999.000000
103	Liam	Smith	1700.000000
104	Olivia	Johnson	1800.000000
105	Noah	Brown	1900.000000
106	Ava	Davis	2000.000000
107	William	Wilson	2100.000000
108	Sophia	Moore	2200.000000
109	James	Taylor	2300.000000
110	Isabella	Anderson	0.000000

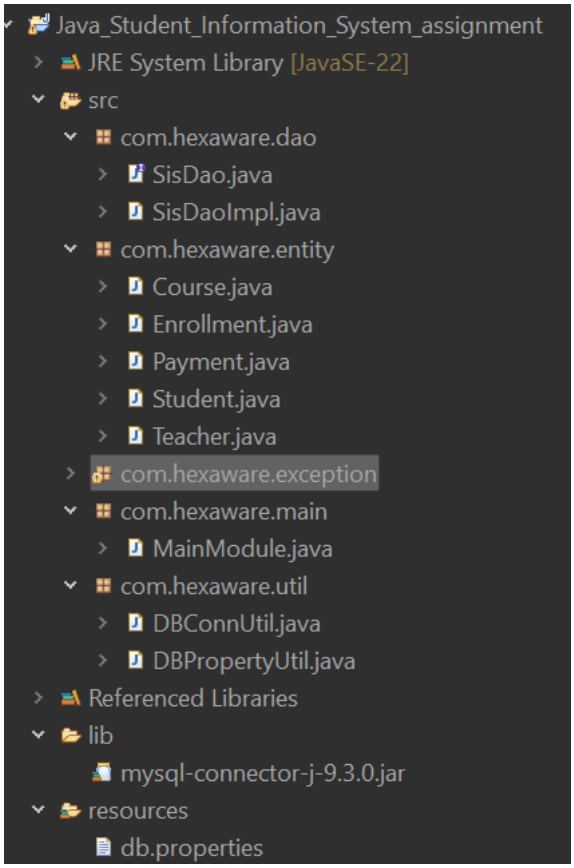
Java

Task 1: Define Classes :

Student class with the following attributes:

- Student ID
- First Name
- Last Name
- Date of Birth
- Email
- Phone Number

```java
public class Student {
    private int studentId;
    private String firstName;
    private String lastName;
    private LocalDate dateOfBirth;
    private String email;
    private String phoneNumber;
    private List<Course> enrolledCourses;
    private List<Payment> paymentHistory;
```



Course class with the following attributes:

- Course ID
- Course Name
- Course Code
- Instructor Name

Enrollment class to represent the relationship between students and courses.

It should have attributes:

- Enrollment ID
- Student ID (reference to a Student)
- Course ID (reference to a Course)
- Enrollment Date

Teacher class with the following attributes:

- Teacher ID
- First Name
- Last Name
- Email

Payment class with the following attributes:

- Payment ID
- Student ID (reference to a Student)
- Amount
- Payment Date

```java
public class Payment {
    private int paymentId;
    private int studentId;
    private double amount;
    private LocalDate paymentDate;
```

Task 2: Implement Constructors

Student Class Constructor

In the Student class, you need to create a constructor that initializes the attributes of a student when an instance of the Student class is created.

```java
// Constructor
public Student(int studentId, String firstName, String lastName,
        LocalDate dateOfBirth, String email, String phoneNumber) {
    this.studentId = studentId;
    this.firstName = firstName;
    this.lastName = lastName;
    this.dateOfBirth = dateOfBirth;
    this.email = email;
    this.phoneNumber = phoneNumber;
    this.enrolledCourses = new ArrayList<>();
    this.paymentHistory = new ArrayList<>();
}
```

SIS Class Constructor

If you have a class that represents the Student Information System itself (e.g., SIS class), you may also implement a constructor for it. This constructor can be used to set up any initial configuration for the SIS. Repeat the above process for each class Course, Enrollment, Teacher, Payment by defining constructors that initialize their respective attributes.

Task 3: Implement Methods

Student Class:

- EnrollInCourse(course: Course): Enrolls the student in a course.
- UpdateStudentInfo(firstName: string, lastName: string, dateOfBirth: DateTime, email: string,
- phoneNumber: string): Updates the student's information.
- MakePayment(amount: decimal, paymentDate: DateTime): Records a payment made by the
- student.
- DisplayStudentInfo(): Displays detailed information about the student.
- GetEnrolledCourses(): Retrieves a list of courses in which the student is enrolled.
- GetPaymentHistory(): Retrieves a list of payment records for the student.

```java
// Enroll student in a course
public void enrollInCourse(Course course) {
    enrolledCourses.add(course);
}

// Update student information
public void updateStudentInfo(String firstName, String lastName, LocalDate dateOfBirth, String email, String phoneNumber) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.dateOfBirth = dateOfBirth;
    this.email = email;
    this.phoneNumber = phoneNumber;
}

// Record a payment
public void makePayment(double amount, LocalDate paymentDate) {
    Payment payment = new Payment(paymentHistory.size() + 1, this.studentId, amount, paymentDate);
    paymentHistory.add(payment);
}
```

Course Class:

- AssignTeacher(teacher: Teacher): Assigns a teacher to the course.
- UpdateCourseInfo(courseCode: string, courseName: string, instructor: string): Updates course
- information.
- DisplayCourseInfo(): Displays detailed information about the course.
- GetEnrollments(): Retrieves a list of student enrollments for the course.
- GetTeacher(): Retrieves the assigned teacher for the course.

```java
// Assign a teacher
public void assignTeacher(Teacher teacher) {
    this.teacher = teacher;
}

// Update course info
public void updateCourseInfo(String courseCode, String courseName, String instructorName) {
    this.courseCode = courseCode;
    this.courseName = courseName;
    this.instructorName = instructorName;
}

// Display course info
public void displayCourseInfo() {
    System.out.println("Course ID: " + courseId);
    System.out.println("Course Name: " + courseName);
    System.out.println("Course Code: " + courseCode);
    System.out.println("Instructor Name: " + instructorName);
}

// Get list of enrollments
public List<Enrollment> getEnrollments() {
    return enrollments;
}
```

Teacher Class:

- UpdateTeacherInfo(name: string, email: string, expertise: string): Updates teacher information.
- DisplayTeacherInfo(): Displays detailed information about the teacher.
- GetAssignedCourses(): Retrieves a list of courses assigned to the teacher.

```java
// Update teacher info
public void updateTeacherInfo(String firstName, String lastName, String email) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
}

// Display teacher info
public void displayTeacherInfo() {
    System.out.println("Teacher ID: " + teacherId);
    System.out.println("Name: " + firstName + " " + lastName);
    System.out.println("Email: " + email);
}

// Get assigned courses
public List<Course> getAssignedCourses() {
    return assignedCourses;
}
```

Enrollment Class:

- GetStudent(): Retrieves the student associated with the enrollment.
- **GetCourse(): Retrieves the course associated with the enrollment.**

Payment Class:

- GetStudent(): Retrieves the student associated with the payment.
- GetPaymentAmount(): Retrieves the payment amount.
- GetPaymentDate(): Retrieves the payment date.

```
public double getAmount() {
    return amount;
```

```
public LocalDate getPaymentDate() {
    return paymentDate;
}
```

Task 4: Exceptions handling and Custom Exceptions

Create Custom Exception Classes

- DuplicateEnrollmentException: Thrown when a student is already enrolled in a course and tries to enroll again. This exception can be used in the EnrollStudentInCourse method.
- CourseNotFoundException: Thrown when a course does not exist in the system, and you
- attempt to perform operations on it (e.g., enrolling a student or assigning a teacher).
- StudentNotFoundException: Thrown when a student does not exist in the system, and you attempt to perform operations on the student (e.g., enrolling in a course, making a payment).
- TeacherNotFoundException: Thrown when a teacher does not exist in the system, and you attempt to assign them to a course.
- PaymentValidationException: Thrown when there is an issue with payment validation, such as an invalid payment amount or payment date.
- InvalidStudentDataException: Thrown when data provided for creating or updating a student is
- invalid (e.g., invalid date of birth or email format).
- InvalidCourseDataException: Thrown when data provided for creating or updating a course is
- invalid (e.g., invalid course code or instructor name).
- InvalidEnrollmentDataException: Thrown when data provided for creating an enrollment is
- invalid (e.g., missing student or course references).
- InvalidTeacherDataException: Thrown when data provided for creating or updating a teacher is
- invalid (e.g., missing name or email).
- InsufficientFundsException: Thrown when a student attempts to enroll in a course but does not
- have enough funds to make the payment.

Task 5: Collections

Implement Collections:

Implement relationships between classes using appropriate data structures (e.g., lists or dictionaries) to maintain associations between students, courses, enrollments, teachers, and payments. These relationships are essential for the Student Information System (SIS) to track and manage student enrollments, teacher assignments, and payments accurately.

Define Class-Level Data Structures

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

Student Class:

```
// Get list of enrolled courses
public List<Course> getEnrolledCourses() {
    return enrolledCourses;
}
```

Course Class:

```
// Get list of enrollments
public List<Enrollment> getEnrollments() {
    return enrollments;
}
```

Enrollment Class:

Include properties to hold references to both the Student and Course objects. Example: Student Student { get; set; } and Course Course { get; set; }

Teacher Class:

```
// Get assigned courses
public List<Course> getAssignedCourses() {
    return assignedCourses;
}
```

Payment Class:

```java
private List<Course> enrolledCourses;
private List<Payment> paymentHistory;
```

Task 6: Create Methods for Managing Relationships

- **AddEnrollment(student, course, enrollmentDate):**

```java
public void addEnrollment(Student student, Course course, Date enrollmentDate) throws DuplicateEnrollmentException, InvalidE
    for (Enrollment enrollment : enrollments) {
        if (enrollment.getStudent().getStudentId() == student.getStudentId() && enrollment.getCourse().getCourseId() == cour
            throw new DuplicateEnrollmentException("Student " + student.getFirstName() + " is already enrolled in this cours
        }
    }
    Enrollment newEnrollment = new Enrollment(student, course, new java.sql.Date(enrollmentDate.getTime()));
    enrollments.add(newEnrollment);
    student.enrollInCourse(course);
    course.getEnrollments().add(newEnrollment);
    System.out.println("Enrollment added for student " + student.getFirstName() + " in course " + course.getCourseName());
}
```

- **AssignCourseToTeacher(course, teacher):**

```java
public void assignCourseToTeacher(Course course, Teacher teacher) throws TeacherNotFoundException {
    if (teacher == null) {
        throw new TeacherNotFoundException("Teacher not found.");
    }
    course.assignTeacher(teacher);
    teacher.getAssignedCourses().add(course);
    System.out.println("Course " + course.getCourseName() + " has been assigned to teacher " + teacher.getFirstName());
}
```

- **AddPayment(student, amount, paymentDate):**

```java
public void addPayment(Student student, double amount, Date paymentDate) throws PaymentValidationException {
    if (amount <= 0) {
        throw new PaymentValidationException("Payment amount must be greater than zero.");
    }
    Payment payment = new Payment(payments.size() + 1, student.getStudentId(), amount, paymentDate);
    payments.add(payment);
    student.makePayment(amount, paymentDate);
    System.out.println("Payment of " + amount + " has been recorded for student " + student.getFirstName());
}
```

- **GetEnrollmentsForStudent(student):**

```java
public List<Enrollment> getEnrollmentsForStudent(Student student) {
    List<Enrollment> studentEnrollments = new ArrayList<>();
    for (Enrollment enrollment : enrollments) {
        if (enrollment.getStudent().getStudentId() == student.getStudentId()) {
            studentEnrollments.add(enrollment);
        }
    }
    return studentEnrollments;
}
```

- **GetCoursesForTeacher(teacher):**

```java
public void assignCourseToTeacher(Course course, Teacher teacher) throws TeacherNotFoundException {
    if (teacher == null) {
        throw new TeacherNotFoundException("Teacher not found.");
    }
    course.assignTeacher(teacher);
    teacher.getAssignedCourses().add(course);
    System.out.println("Course " + course.getCourseName() + " has been assigned to teacher " + teacher.getFirstName());
}
```

Task 7: Database Connectivity

Database Initialization:

```
☑ Database connected successfully!
Welcome to the Student Information System
1. Add a student
2. View all students
3. Enroll a student in a course
4. View students enrolled in a course
5. Exit
Enter your choice:
```

Data Retrieval:

```
===== Student Information System (SIS) =====
1. Add Student
2. View All Students
3. Enroll Student to Course
4. Add Course
5. Add Teacher
6. Assign Teacher to Course
7. Make Payment
8. View Student Payment History
9. Generate Enrollment Report
0. Exit
Enter your choice: 2
```

```
--- View All Students ---
Student ID: 1
Name: John Doe
DOB: 2000-05-15
Email: john.doe@example.com
Phone: 1234567890
--------------------------------
Student ID: 2
Name: Jane Smith
DOB: 1999-08-22
Email: jane.smith@example.com
Phone: 2345678901
--------------------------------
```

Data Insertion and Updating:

```
--- Add Course ---
Enter Course Name: Introduction to programming
Enter Course Code: 7
Enter Instructor Name: Zoro
Course added successfully!
☑ Course added successfully!
```

```
Enter your choice: 1
Enter student details:
Student ID: 7
First Name: johnn
Last Name: doe
Date of Birth (YYYY-MM-DD): 1995-08-15
Email: johnn.doe@example.com
Phone Number: 1234567890
☑ Student added successfully.
```

Transaction Management:

Implement methods for handling database transactions when enrolling students, assigning teachers, or recording payments. Transactions should be atomic and maintain data integrity. Use database transactions to ensure that multiple related operations either all succeed or all fail. Implement error handling and rollback mechanisms in case of transaction failures.

Task 8: Student Enrollment

Create a new student record in the database.

- Enroll John in the specified courses by creating enrollment records in the database.

```
--- Add Student ---
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!
```

```
Enter your choice: 3
Enter student ID to enroll: 7
Enter course ID to enroll in: 1
Enter enrollment date (YYYY-MM-DD): 2025-04-19
Enrollment added successfully.
Student enrolled in the course successfully.
```

Task 9: Teacher Assignment

Teacher's Details:

```
===== Student Information System (SIS) =====
1. Add Student
2. View All Students
3. Enroll Student to Course
4. Add Course
5. Add Teacher
6. Assign Teacher to Course
7. Make Payment
8. View Student Payment History
9. Generate Enrollment Report
0. Exit
Enter your choice: 6
```

```
--- Assign Teacher to Course ---
Enter Course ID: 8
Enter Teacher ID: 6
Teacher assigned to course successfully!
```

Task 10: Payment Record

.

Jane Johnson's details:

- Student ID: 101
- Payment Amount: $500.00
- Payment Date: 2023-04-10

  .

```
===== Student Information System (SIS) =====
1. Add Student
2. View All Students
3. Enroll Student to Course
4. Add Course
5. Add Teacher
6. Assign Teacher to Course
7. Make Payment
8. View Student Payment History
9. Generate Enrollment Report
0. Exit
Enter your choice: 7
```

```
--- Make Payment ---
Enter Student ID: 9
Enter Amount: 500
Enter Payment Date (yyyy-mm-dd): -2023-04-10
Payment recorded successfully!
```

Task 11: Enrollment Report Generation

```
===== Student Information System (SIS) =====
1. Add Student
2. View All Students
3. Enroll Student to Course
4. Add Course
5. Add Teacher
6. Assign Teacher to Course
7. Make Payment
8. View Student Payment History
9. Generate Enrollment Report
0. Exit
Enter your choice: 9
```

```
--- Generate Enrollment Report ---
Enter Course Id: 3

===== Enrollment Report =====
Course Name: Computer Science 101

Enrolled Students:
Student ID: 3
Name        : Alice Johnson
```