Student Information System (SIS)

Instructions:

- Submitting assignments should be a single file or through git hub link shared with trainer and hexavarsity.
- Each assignment builds upon the previous one, and by the end, you will have a comprehensive application implemented in Java/C#/Python with a strong focus on SQL schema design, control flow statements, loops, arrays, collections, and database interaction.
- Follow object-oriented principles throughout the Java programming assignments. Use classes and objects to model real-world entities, encapsulate data and behavior, and ensure code reusability.
- Throw user defined exception from method and handle in the main method.
- The following Directory structure is to be followed in the application.
    - **entity/model**
        - Create entity classes in this package. All entity class should not have any business logic.
    - **dao**
        - Create Service Provider interface/abstract class to showcase functionalities.
        - Create the implementation class for the above interface/abstract class with db interaction.
    - **exception**
        - Create user defined exceptions in this package and handle exceptions whenever needed.
    - **util**
        - Create a DBPropertyUtil class with a static function which takes property file name as parameter and returns connection string.
        - Create a DBConnUtil class which holds static method which takes connection string as parameter file and returns connection object.
    - **main**
        - Create a class MainModule and demonstrate the functionalities in a menu driven application.

In this assignment, you will work with a simplified Student Information System (SIS) database. The SIS database contains information about students, courses, and enrollments. Your task is to perform various SQL operations on this database to retrieve and manipulate data.

Database Tables:

The SIS database consists of the following tables:

1. **Students**
    - student_id (Primary Key)
    - first_name
    - last_name
    - date_of_birth
    - email
    - phone_number
2. **Courses**

- course_id (Primary Key)
- course_name
- credits
- teacher_id (Foreign Key)

3. **Enrollments**
   - enrollment_id (Primary Key)
   - student_id (Foreign Key)
   - course_id (Foreign Key)
   - enrollment_date
4. **Teacher**
   - teacher_id (Primary Key)
   - first_name
   - last_name
   - email
5. **Payments**
   - payment_id (Primary Key)
   - student_id (Foreign Key)
   - amount
   - payment_date

Task 1. Database Design:

1. Create the database named "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
3. Create an ERD (Entity Relationship Diagram) for the database.
4. Create appropriate Primary Key and Foreign Key constraints for referential integrity.
5. Insert at least 10 sample records into each of the following tables.
   i. Students
   ii. Courses
   iii. Enrollments
   iv. Teacher
   v. Payments

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
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.
3. Update the email address of a specific teacher in the "Teacher" table. Choose any teacher and modify their email address.
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.
5. Update the "Courses" table to assign a specific teacher to a course. Choose any course and teacher from the respective tables.
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.
7. Update the payment amount for a specific payment record in the "Payments" table. Choose any payment record and modify the payment amount.

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.
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.
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.
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.
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.
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.
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.
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.

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.
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.

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.
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.
3. Retrieve a list of courses with the highest number of enrollments. Use subqueries to find the course(s) with the maximum enrollment count.
4. Calculate the total payments made to courses taught by each teacher. Use subqueries to sum payments for each teacher's courses.
5. Identify students who are enrolled in all available courses. Use subqueries to compare a student's enrollments with the total number of courses.
6. Retrieve the names of teachers who have not been assigned to any courses. Use subqueries to find teachers with no course assignments.
7. Calculate the average age of all students. Use subqueries to calculate the age of each student based on their date of birth.
8. Identify courses with no enrollments. Use subqueries to find courses without enrollment records.
9. Calculate the total payments made by each student for each course they are enrolled in. Use subqueries and aggregate functions to sum payments.
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.
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.
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.
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.

Student Information System (SIS)

Implement OOPs

A Student Information System (SIS) manages information about students, courses, student enrollments, teachers, and payments. Each student can enroll in multiple courses, each course can have multiple students, each course is taught by a teacher, and students make payments for their courses. Students have attributes such as name, date of birth, email, and phone number. Courses have attributes such as course name, course code, and instructor name. Enrollments track which students are enrolled in which courses. Teachers have attributes such as names and email. Payments track the amount and date of payments made by students.

Task 1: Define Classes

Define the following classes based on the domain description:

Student class with the following attributes:

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

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

Task 2: Implement Constructors

Implement constructors for each class to initialize their attributes. Constructors are special methods that are called when an object of a class is created. They are used to set initial values for the attributes of the class. Below are detailed instructions on how to implement constructors for each class in your Student Information System (SIS) assignment:

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

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

Implement methods in your classes to perform various operations related to the Student Information System (SIS). These methods will allow you to interact with and manipulate data within your system. Below are detailed instructions on how to implement methods in each class:

Implement the following methods in the appropriate classes:

Student Class:

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

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.

Enrollment Class:

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

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.

Payment Class:

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

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

- EnrollStudentInCourse(student: Student, course: Course): Enrolls a student in a course.
- AssignTeacherToCourse(teacher: Teacher, course: Course): Assigns a teacher to a course.
- RecordPayment(student: Student, amount: decimal, paymentDate: DateTime): Records a payment made by a student.
- GenerateEnrollmentReport(course: Course): Generates a report of students enrolled in a specific course.
- GeneratePaymentReport(student: Student): Generates a report of payments made by a specific student.
- CalculateCourseStatistics(course: Course): Calculates statistics for a specific course, such as the number of enrollments and total payments.

Use the Methods

In your driver program or any part of your code where you want to perform actions related to the Student Information System, create instances of your classes, and use the methods you've implemented.

Repeat this process for using other methods you've implemented in your classes and the SIS class.

Task 4: Exceptions handling and Custom Exceptions

Implementing custom exceptions allows you to define and throw exceptions tailored to specific situations or business logic requirements.

Create Custom Exception Classes

You'll need to create custom exception classes that are inherited from the System.Exception class or one of its derived classes (e.g., System.ApplicationException). These custom exception classes will allow you to encapsulate specific error scenarios and provide meaningful error messages.

Throw Custom Exceptions

In your code, you can throw custom exceptions when specific conditions or business logic rules are violated. To throw a custom exception, use the throw keyword followed by an instance of your custom exception class.

- **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:

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

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

Course Class:

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

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

Enrollment Class:

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

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

Teacher Class:

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

Example: List<Course> AssignedCourses { get; set; }

Payment Class:

Include a property to hold a reference to the Student object.

Example: Student Student { get; set; }

Update Constructor(s)

In the constructors of your classes, initialize the list or collection properties to create empty collections when an object is instantiated.

Repeat this for the Course, Teacher, and Payment classes, where applicable.

Task 6: Create Methods for Managing Relationships

To add, remove, or retrieve related objects, you should create methods within your SIS class or each relevant class.

- **AddEnrollment**(student, course, enrollmentDate): In the SIS class, create a method that adds an enrollment to both the Student's and Course's enrollment lists. Ensure the Enrollment object references the correct Student and Course.
- **AssignCourseToTeacher**(course, teacher): In the SIS class, create a method to assign a course to a teacher. Add the course to the teacher's AssignedCourses list.
- **AddPayment**(student, amount, paymentDate): In the SIS class, create a method that adds a payment to the Student's payment history. Ensure the Payment object references the correct Student.
- **GetEnrollmentsForStudent**(student): In the SIS class, create a method to retrieve all enrollments for a specific student.
- **GetCoursesForTeacher**(teacher): In the SIS class, create a method to retrieve all courses assigned to a specific teacher.

Create a Driver Program

A driver program (also known as a test program or main program) is essential for testing and demonstrating the functionality of your classes and methods within your Student Information System (SIS) assignment. In this task, you will create a console application that serves as the entry point for your SIS and allows you to interact with and test your implemented classes and methods.

Add References to Your SIS Classes

Ensure that your SIS classes (Student, Course, Enrollment, Teacher, Payment) and the SIS class (if you have one to manage interactions) are defined in separate files within your project or are referenced properly.

If you have defined these classes in separate files, make sure to include using statements in your driver program to access them:

Implement the Main Method

In the console application, the Main method serves as the entry point for your program. This is where you will create instances of your classes, call methods, and interact with your Student Information System.

In the Main method, you create instances of your classes (e.g., Student, Course, and SIS) and then interact with your Student Information System by calling methods and handling exceptions.

Task 7: Database Connectivity

Database Initialization:

Implement a method that initializes a database connection and creates tables for storing student, course, enrollment, teacher, and payment information. Create SQL scripts or use code-first migration to create tables with appropriate schemas for your SIS.

Data Retrieval:

Implement methods to retrieve data from the database. Users should be able to request information about students, courses, enrollments, teachers, or payments. Ensure that the data retrieval methods handle exceptions and edge cases gracefully.

Data Insertion and Updating:

Implement methods to insert new data (e.g., enrollments, payments) into the database and update existing data (e.g., student information). Use methods to perform data insertion and updating. Implement validation checks to ensure data integrity and handle any errors during these operations.

Transaction Management:

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

Dynamic Query Builder:

Implement a dynamic query builder that allows users to construct and execute custom SQL queries to retrieve specific data from the database. Users should be able to specify columns, conditions, and sorting criteria. Create a query builder method that dynamically generates SQL queries based on user input. Implement parameterization and sanitation of user inputs to prevent SQL injection.

Task 8: Student Enrollment

In this task, a new student, John Doe, is enrolling in the SIS. The system needs to record John's information, including his personal details, and enroll him in a few courses. Database connectivity is required to store this information.

John Doe's details:

- First Name: John
- Last Name: Doe
- Date of Birth: 1995-08-15
- Email: john.doe@example.com
- Phone Number: 123-456-7890

John is enrolling in the following courses:

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

The system should perform the following tasks:

- Create a new student record in the database.
- Enroll John in the specified courses by creating enrollment records in the database.

Task 9: Teacher Assignment

In this task, a new teacher, Sarah Smith, is assigned to teach a course. The system needs to update the course record to reflect the teacher assignment.

Teacher's Details:

- Name: Sarah Smith
- Email: sarah.smith@example.com
- Expertise: Computer Science

Course to be assigned:

- Course Name: Advanced Database Management
- Course Code: CS302

The system should perform the following tasks:

- Retrieve the course record from the database based on the course code.
- Assign Sarah Smith as the instructor for the course.
- Update the course record in the database with the new instructor information.

Task 10: Payment Record

In this task, a student, Jane Johnson, makes a payment for her enrolled courses. The system needs to record this payment in the database.

Jane Johnson's details:

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

The system should perform the following tasks:

- Retrieve Jane Johnson's student record from the database based on her student ID.
- Record the payment information in the database, associating it with Jane's student record.
- Update Jane's outstanding balance in the database based on the payment amount.

Task 11: Enrollment Report Generation

In this task, an administrator requests an enrollment report for a specific course, "Computer Science 101." The system needs to retrieve enrollment information from the database and generate a report.

Course to generate the report for:

- Course Name: Computer Science 101

The system should perform the following tasks:

- Retrieve enrollment records from the database for the specified course.
- Generate an enrollment report listing all students enrolled in Computer Science 101.
- Display or save the report for the administrator.