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 "SIS"

```
Create database SISDB;

            use SISDB;
```

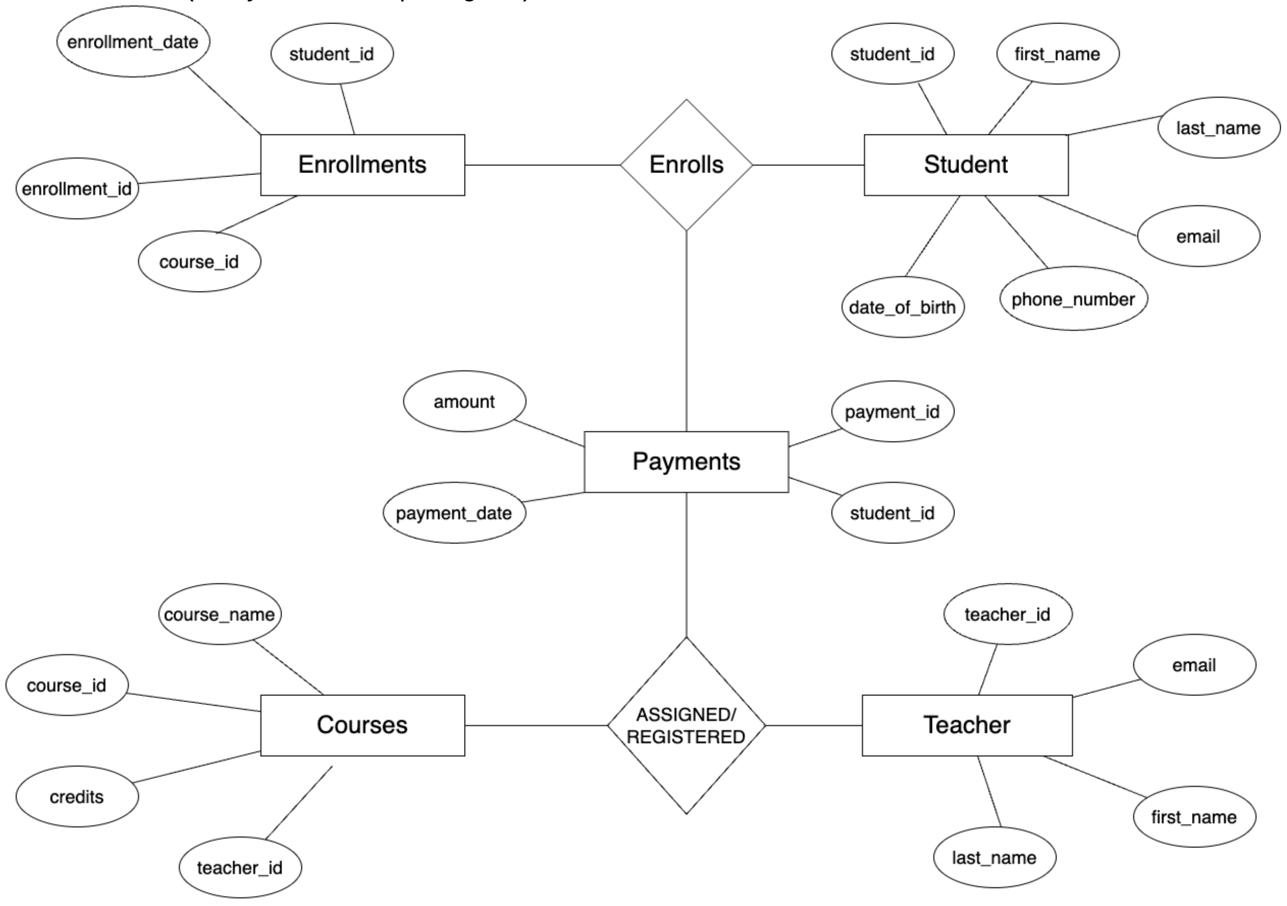
2. Define the schema for the Students, Courses, Enrollments, Teacher, and Payments tables based on the provided schema. Write SQL scripts to create the mentioned tables with appropriate data types, constraints, and relationships.
   a. Students
   b. Courses
   c. Enrollments
   d. Teacher
   e. Payments

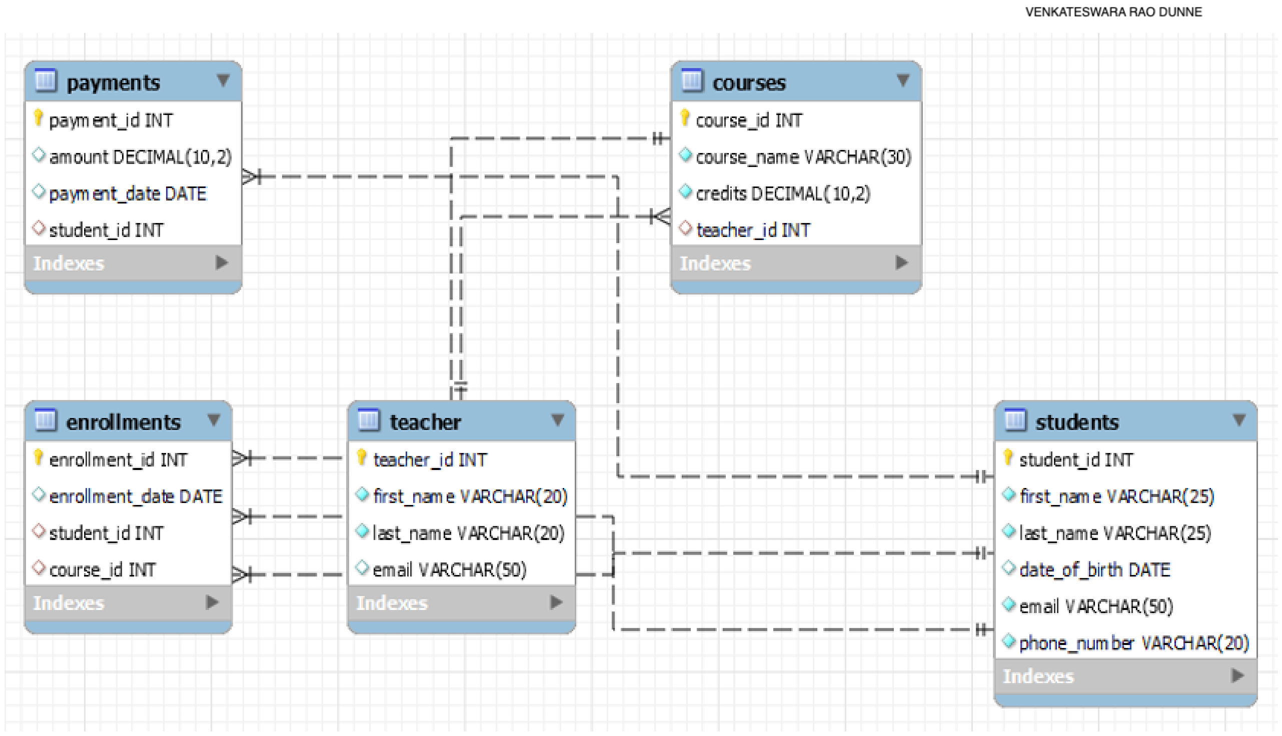
4. Create appropriate Primary Key and Foreign Key constraints for referential integrity.

```
create table Students(
 student_id int primary key auto_increment unique,
 first_name varchar(25) not null,
 last_name varchar(25) not null,
 date_of_birth date,
 email varchar(50) not null,
 phone_number varchar(20) not null
);



create table Teacher(
            teacher_id int primary key not null unique,
            first_name varchar(20) not null,
            last_name varchar(20) not null,
            email varchar(50)
);
```

```
create table Courses(
        course_id int primary key,
        course_name varchar(30) not null,
        credits decimal(10,2) not null,
        teacher_id int,
        foreign key(teacher_id) references Teacher(teacher_id)
);



create table Enrollments(
        enrollment_id int primary key,
        enrollment_date Date,
        student_id int,
        course_id int,
        foreign key(student_id) references Students(student_id),
        foreign key(course_id) references Courses(course_id)
);



create table Payments(
        payment_id int primary key,
        amount decimal(10,2),
        payment_date date,
        student_id int,
        foreign key(student_id) references Students(student_id)
);
```

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





5. Insert at least 10 sample records into each of the following tables.
   i. Students
   ii. Courses
   iii. Enrollments
   iv. Teacher
   v. Payments

```
insert into Students values
 (1,'pavan','vardhan','2000-03-21','pavanvardhan@gmail.com',9876123456),
 (2,'palla','rao','2001-05-12','pallarao@gmail.com',985736373439),
 (3,'venkat','dunne','2000-05-06','venkat.dunne@gmail.com',9954459999),
(4,'priya','surampalli','2001-12-06','priyasurampalli@gmail.com',8765424609),
 (5,'uma','srivalli','2000-03-12','umasirivalli@gmail.com',98756377373)
;


insert into Teacher values
 (11,'Anushka','s','anushka.s@gmail.com'),
 (12,'deepika','p','deepika.p@gmail.com'),
 (13,'shruthi','h','shruthi.h@gmail.com'),
 (14,'rashmika','d','rashmika.d@gmail.com'),
 (15,'alia','b','alia.b@gmail.com')
;


insert into Courses values
 (101,'NextJS',3.6,11),
 (102,'React',4.0,12),
 (103,'NodeJS',3.6,13),
 (104,'Anjular',4.0,14),
 (105,'MongoDB',3.9,15)
;


insert into Enrollments values
 (1005,'2023-08-11',1,105),
 (1009,'2023-07-10',2,104),
(1008,'2023-07-06',3,103),
 (1006,'2023-07-18',4,102),
 (1007,'2023-07-19',5,101)
;
insert into Payments values
 (001,435.00,'2023-11-21',1),
 (002,557.00,'2023-11-21',2),
 (003,876.00,'2023-11-21',3),
 (004,982.00,'2023-11-21',4),
 (005,345.00,'2023-11-21',5)
;
```

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(first_name,last_name,date_of_birth,email,phone_number) values('John','Doe','1998-08-15','john.doe@example.com',1234567890);
select * from Students;
```

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(1004,1, 101,'2023-12-07');
select * from Enrollments;
```

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 = 'something@gmail.com'
where teacher_id = 12;
 select * from Teacher;
```

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 = 2 AND course_id = 102;

select * from Enrollments;
```

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 = 12

where course_id = 101;

select * from Courses;
```

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 = 2;

///delete from Students where Student_id = 2;

select * from Students;

select * from Enrollments;
```

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 = 125

where payment_id = 001;

select * from Payments;
```

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.

```
select Students.student_id,concat(Students.first_name,' ',Students.last_name) as Name, count(Payments.amount) as total payments

from Students left join Payments on Students.student_id = Payments.payment_id

where Students.student_id = 2 ;
```

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.

```
select Courses.course_name,count(Courses.course_id) as Total

from Courses left join Enrollments

on Courses.course_id = Enrollments.course_id

group by Courses.course_id, Courses.course_name;
```

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.

```
select concat(Students.first_name,' ',Students.last_name) as Name

from Students left join Enrollments

on Students.student_id = Enrollments.student_id

where Enrollments.Student_id is null;
```

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 Students.first_name, Students.last_name, Courses.course_name

from Students

join Enrollments on Students.student_id = Enrollments.student_id

join Courses on Enrollments.course_id = Courses.course_id;
```

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 concat(Teacher.first_name,' ',Teacher.last_name) as TeacherName, Courses.course_namefrom Teacher
join Courses

on Teacher.teacher_id = Courses.teacher_id;
```

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 concat(Students.first_name,' ',Students.last_name) asStudentName,Enrollments.enrollment_date

from Students

join Enrollments on Students.student_id = Enrollments.student_id

join Courses on Enrollments.course_id = Courses.course_id;
```

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 concat(Students.first_name,' ',Students.last_name) as studentname

from Students

left join Payments on Students.student_id = Payments.student_id

where Payments.amount is null;
```

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 Courses.course_name from Courses

left join Enrollments on Courses.course_id = Enrollments.course_id

where Enrollments.enrollment_id is null;
```

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 e.student_id
from Enrollments e
join Enrollments e1 on e.student_id = e1.student_id and e.course_id<> e1.course_id
join Students s on e.student_id = s.student_id;
```

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 concat(Teacher.first_name,' ',Teacher.last_name) as TeacherName
from Teacher left join Courses
on Teacher.teacher_id = Courses.teacher_id
where Courses.course_id is null;
```

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 Courses.course_id, Courses.course_name, avg(EnrollmentCount.student_count) as averagestudentsenroll
from Courses
left join (select course_id, COUNT(distinct student_id) as student_count
from Enrollments group by course_id) as EnrollmentCount
on Courses.course_id = EnrollmentCount.course_id
group by Courses.course_id, Courses.course_name;
```

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 student_idfrom Payments
where amount = (select MAX(amount)
from Payments
);
```

);

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

```
select student_idfrom Payments

where amount = (select MAX(amount)

from Payments

);
```

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

```
select Teacher.teacher_id,

Teacher.first_name,

coalesce(SUM(Payments.amount), 0) as total_payments

from Teacher

left join Courses on Teacher.teacher_id = Courses.teacher_id

left join Enrollments on Courses.course_id = Enrollments.course_id

left join Payments on Enrollments.student_id = Payments.student_id

group by Teacher.teacher_id, Teacher.first_name;
```

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

```
select student_id,first_name,last_name
from Students

where not exists (

        select course_id

        from Courses

        where not exists (

        select 1

        from Enrollments

        where Enrollments.student_id = Students.student_id

        and Enrollments.course_id = Courses.course_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 teacher_id, first_name, last_name
from Teacher
where not exists (
        select 1
        from Courses
        where Courses.teacher_id = Teacher.teacher_id
);
```

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(DATEDIFF(CURDATE(), date_of_birth) / 365) as average_age
from Students;
```

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

```
select course_id, course_name
from Courses
where not exists (select 1from Enrollments
 where Enrollments.course_id = Courses.course_id
);
```

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 Students.student_id, Students.first_name, Students.last_name,Courses.course_id, Courses.course_name,
coalesce(SUM(Payments.amount), 0) as total_payments
from Students
join Enrollments on Students.student_id = Enrollments.student_id
join Courses on Enrollments.course_id = Courses.course_id
left join Payments on Enrollments.student_id = Payments.student_id
group by Students.student_id, Students.first_name, Students.last_name, Courses.course_id,
Courses.course_name;
```

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, student_name
FROM (
        SELECT student_id, student_name, COUNT(payment_id) AS payment_count
        FROM Students
        LEFT JOIN Payments ON Students.student_id = Payments.student_id
        GROUP BY Students.student_id, Students.student_name
) AS StudentPaymentCounts
WHERE payment_count> 1;
```

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 Students.student_id, Students.first_name, Students.last_name, coalesce(SUM(Payments.amount), 0) as
total_payments
from Students
left join Payments on Students.student_id = Payments.student_id
group by Students.student_id, Students.first_name, Students.first_name;
```

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 Courses.course_id, Courses.course_name, COUNT(Enrollments.student_id) AS student_count
from Courses
left join Enrollments on Courses.course_id = Enrollments.course_id
group by Courses.course_id, Courses.course_name;
```

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 avg(amount) as average_payment_amount
from Payments;
```