**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),

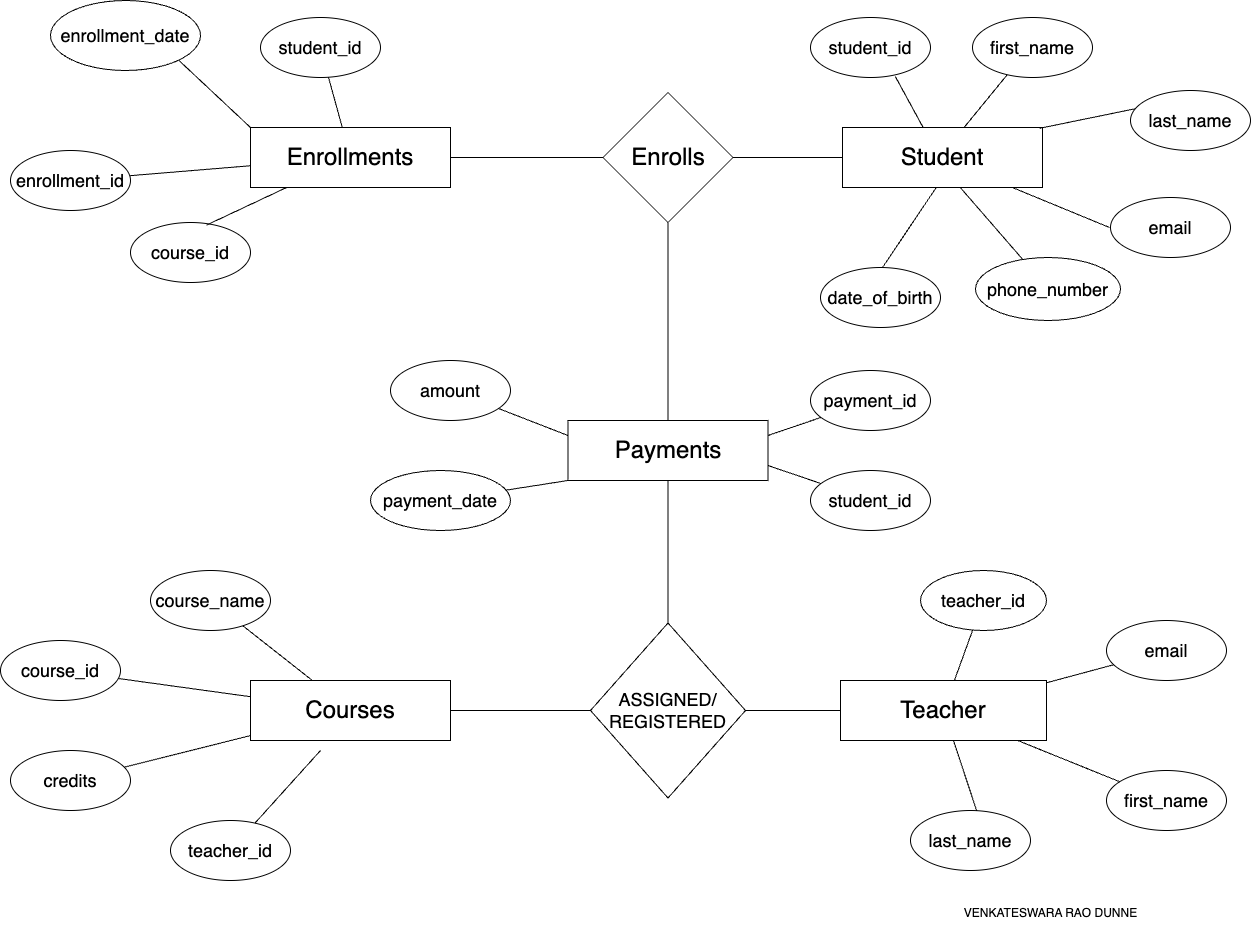
payment\_date date,

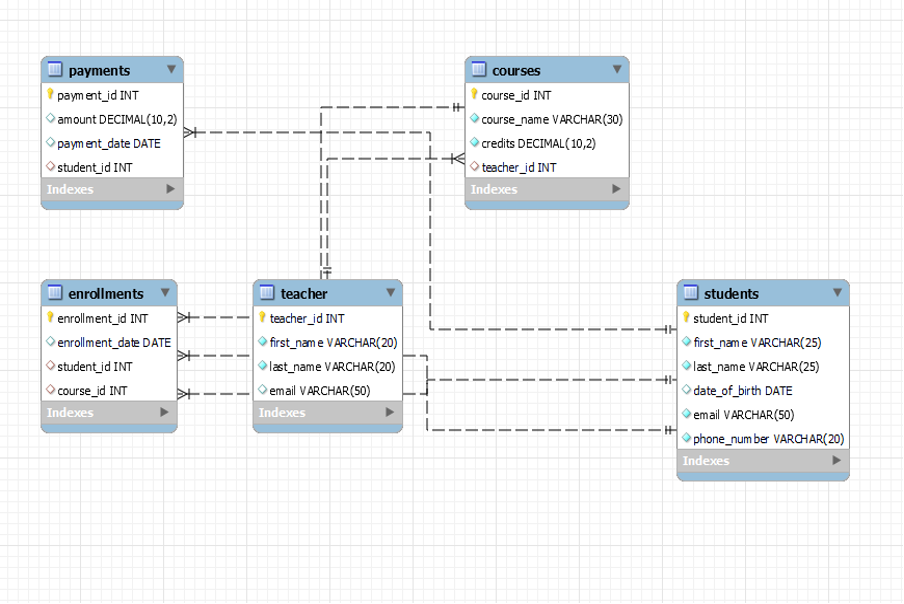
student\_id int,

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;