PROJECT 1

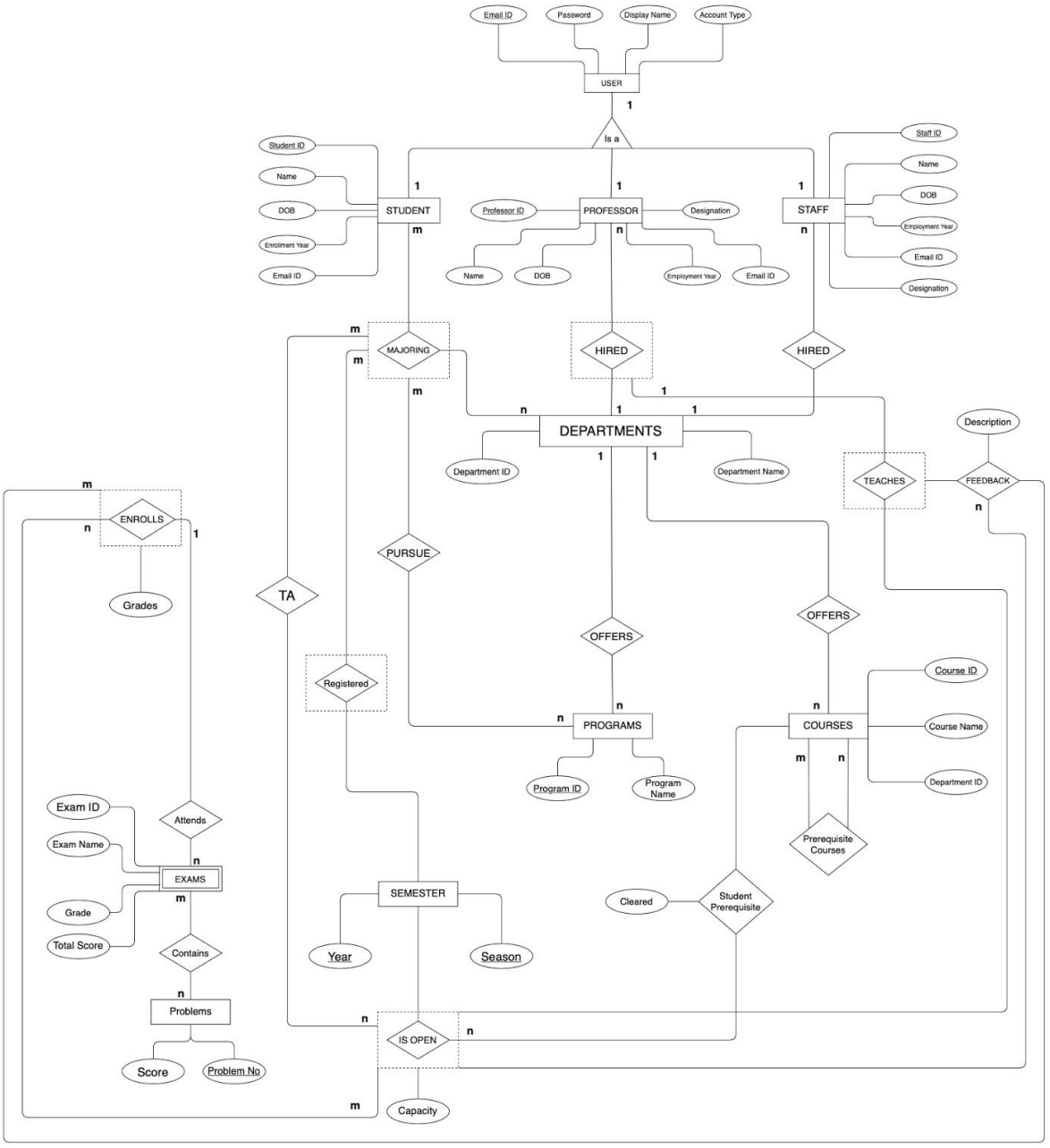
CSE560: Data Models and Query Language

In this project, We have designed and implemented the database schema for TinyHub, which is a course enrollment system. The main functions of TinyHub are the following:

- User Management
- Department Management
- Course Management
- Student Course Management

E/R Schema:

The E/R model for TinyHub is as follows:



User Management:

- A user signs in using his/her email address, display name, password.
- An entity **USER** has four attributes – Email_id, Password, Display_name, User_id.
- Each user has a unique email_id and display_name.
- A User can be Student, Staff or Professor so, three entities **STAFF**, **PROFESSOR** and **STUDENT** are created and made an IS-A relationship with User.
- An entity **STAFF** has staff_id which is referenced from the **User** entity.
- Similarly, an entity **PROFESSOR** has professor_id which is referenced from the **USER** entity.

Department Management:

- An entity **DEPARTMENT** is created and assigned two attributes Dept_Id and Dept_name.
- Since each professor is hired by one Department and one Department can have many professors, the relation between the Department and professor would be **One to Many** and both entities are connected by relation **HIRED_BY**.
- Similarly, Since each staff is hired by one Department and one Department can have multiple staff, the relation between the Department and staff would be **One to Many** and both entities are connected by relation **HIRED_BY**.
- Since each department offers multiple programs and each program can belong to only one department, the relation between the department and programs would be **One to Many** and both entities are connected by relation **Offers**.
- The entity Student and Department are connected by relation **MAJORING_IN**.
- Since one student can major in more than one department and the Department can provide majors to many students, the relation between student and department would be **Many to Many**.
- Since students can pursue different programs and each program has many students, the relation between student and program would be **Many to Many.**

Course Management:

- An entity **COURSE** is created with the attributes course id, course name.
- Since each department offers many courses, the relation between department and course would be **One to Many** and both are connected by relation **OFFERS**.
- A course may contain prerequisite courses. So, the entity course has a recursive relation **PREREQUISITE** to the entity course to get all the prerequisites of any particular course.
- An entity **SEMESTER** is created with attributes year and season.
- Since a course does not have to be opened every semester, the entity course makes relation with the entity semester as **IS_OPEN**. Each semester has multiple courses and each course can be opened in more than one semester, So the relation would be **Many to Many.** This entity Is_open has an attribute capacity which denotes the maximum number of students that the course can accommodate for a particular semester.

- Since one semester can have N courses and one course can be in M semesters, the relation here would be **Many to Many**.
- Any Professor can instruct any course which is opened in a semester.
- Any student can be a TA for multiple courses which are opened in a semester.
- Since each student may register in different semesters and each semester has many students registered, the relation would be **Many to Many** and both entities are connected by relation **IS_REGISTERED**.

Student-Course Management:

- A student can enroll in a course only if that student has registered for a semester in which that particular course is opened. This can be achieved by creating a relation called **ENROLLS** between **IS_REGISTERED** and **IS_OPEN.**
- **ENROLLS** has an attribute grade, which is used to store the letter grade that the student obtains for any course that he enrolled in.
- A student can enroll in a course only if the capacity of the course is not full. This can be achieved by using the attribute **capacity** present in **IS_OPEN**.
- Whenever the student tries to enroll for a course, we check if the course is open in a semester that he registered and also if the capacity is not full.
- Each open course has multiple exams. An entity **EXAM** is created with attributes exam_id, exam_name, and total_score. So the relation would be **One to Many.** The relation **HAS** is made in between Is_open and Exam.
- Each Exam has many problems and each problem has the score. An entity **Problem** is created with an attribute score. The relation between Exam and Problem would be **One to Many.**
- Students can post feedback for the instructor of the course in which they enrolled. This is achieved by creating a relation **FEEDBACK** between Enrolls and Instructs. Feedback has an attribute comment.
- Any student can be a TA for the course which is opened in a semester.

Relational Database Schema:

The Schema consists of following tables:

- USER
- DEPARTMENT
- STUDENT
- PROFESSOR
- STAFF
- COURSES
- PROGRAM
- SEMESTER
- EXAMS
- PROBLEM
- MAJOR_IN
- PURSUE
- REGISTERED

- ISOPEN
- PREREQUISITE
- ENROLL
- TA
- ATTENDS
- FEEDBACK
- TEACHES
- STUDENT_EXAM
- STUDENT_PREREQ

Schema:

1) USERS (email_id, displayname, password, account_type)
   Primary Key – email_id
   Unique Key – display_name

2) DEPARTMENT (department_id, department_name)
   Primary Key – department_id

3) COURSES (courses_id, courses_name, department_id)
   Primary Key – courses_id
   Foreign Key (department_id) references DEPARTMENT(department_id)

4) PROGRAM (program_id, program_name, department_id)
   Primary Key – program_id
   Foreign Key (department_id) references DEPARTMENT (department_id)

5) SEMESTER (years, season)
   Primary Key (years, season)

6) STUDENT (student_id, student_name, DOB, Enroll_year, email_id)
   Primary Key – student_id
   Foreign Key (email_id) references USERS (email_id)

7) PROFESSOR (professor_id, professor_name, DOB, employment_year, email_id, designation)
   Primary Key – professor_id
   Foreign Key (email_id) references USERS (email_id)
   Foreign Key (department_id) references DEPARTMENT (department_id)

8) STAFF (staff_id, staff_name, DOB, employment_year, email_id, designation, department_id)
   Primary_Key (staff_id)
   Foreign Key (email_id) references USERS (email_id)
   Foreign Key (department_id) references DEPARTMENT (department_id)

9) MAJOR_IN (student_id, department_id)
   Foreign Key (student_id) references STUDENT (student_id)

Foreign Key (department_id) references DEPARTMENT (department_id)

10) PURSUE (student_id, department_id, program_id)
Foreign Key (student_id,) references MAJOR_IN (student_id)
Foreign Key (department_id) references PROGRAM (department_id)
Foreign Key (program_id) references PROGRAM (program_id)

11) REGISTERED (student_id, years, season)
Foreign Key (student_id) references MAJOR_IN (student_id)
Foreign Key (years,season) references SEMESTER (years,season)

12) ISOPEN (courses_id, years, season, capacity)
Primary Key (courses_id,years,season)
Foreign Key (years,season) references SEMESTER (years,season)

13) PREREQUISITE (courses_id, prerequisite_id)
Foreign Key (courses_id) references COURSES (courses_id)
Foreign Key (prerequisite_id) references COURSES (courses_id)

14) STUDENT_PREREQ (student_id, courses_id, prerequisite_id, cleared)
Primary Key (courses_id, student_id)
Foreign Key (student_id) references REGISTERED (student_id)
Foreign Key (courses_id) references COURSES (courses_id)
Foreign Key (prerequisite_id) references PREREQUISITE (prerequisite_id)

15) ENROLL (student_id, overall_grade, courses_id, years, season)
Primary Key (student_id, years, season)
Foreign Key (years, season) references ISOPEN (years,season)
Foreign Key (courses_id) references ISOPEN (courses_id)
Foreign Key (student_id) references STUDENT_PREREQ (student_id)
Foreign Key (years) references REGISTERED (years)
Foreign Key (season) references REGISTERED (season)

16) TA (student_id, years, season, courses_id)
Foreign Key (years, season) references ISOPEN (years,season)
Foreign Key (courses_id) references ISOPEN (courses_id)
Foreign Key (student_id) references STUDENT (student_id)

17) TEACHES (professor_id, years, season, courses_id)
Foreign Key (professor_id) references PROFESSOR (professor_id)
Foreign Key (years,season) references ISOPEN (years,season)
Foreign Key (courses_id) references ISOPEN (courses_id)

18) FEEDBACK (student_id, professor_id, courses_id, descriptions)
Foreign Key (professor_id) references TEACHES (professor_id)
Foreign Key (courses_id) references TEACHES (courses_id)
Foreign Key (student_id) references ENROLL (student_id)

19) EXAMS (exams_id, exam_name, courses_id)
    Primary Key – exams_id
    Foreign Key (courses_id) references ISOPEN (courses_id)

20) PROBLEM (problem_id, exams_id)
    Primary Key – problem_id
    Foreign Key (exams_id) references EXAMS (exams_id)

21) ATTENDS (student_id, courses_id, exams_id, overall_grade)
    Foreign Key (courses_id) references ENROLL (courses_id)
    Foreign Key (student_id) references ENROLL (student_id)
    Foreign Key (exams_id) references EXAMS (exams_id)

22) EXAM (student_id, exams_id, score, courses_id, problem_id)
    Foreign Key (courses_id) references ATTENDS (courses_id)
    Foreign Key (student_id) references ATTENDS (student_id)
    Foreign Key (exams_id) references ATTENDS (exams_id)
    Foreign Key (problem_id) references PROBLEM (problem_id)

Schema Justification:

1. In our schema we have Student, Professor and Staff who are basically the users of Tinyhub. We have generalized these using IS A relationship to User relation. (req.2.1.1)
2. Having user passwords isolated from other information about the user makes it more secure. (req.2.1.3)
3. Any user can log in into the system using their email id. Hence we have referenced email id from student, professor and staff relations to user relation.(req. 2.1.6)
4. As Professor and Staff belong to only one department, we have kept a field department_id in their relations referencing department_id in DEPARTMENT relation. (req. 2.2.2 and req. 2.2.3.)
5. As students can major in more than one department, we are storing this information in MAJOR_IN relation. (req. 2.2.5)
6. We have made PROGRAM relation to hold program details along with a field - department_id referencing depart_id in DEPARTMENT. This satisfies the condition that a program can belong to only one department and a department can have many programs. (req. 2.2.4)
7. We have used a PURSUE table which has attributes : student_id referencing student_id and department_id referencing department_id from MAJOR_IN relation. This ensures that students can pursue different programs, but they must major in the department which is offering that program. (req. 2.2.6)
8. We have used IS_OPEN relation to track the courses offered in different semesters. (req. 2.3.2)
9. REGISTERED relation enables students to register in different semesters. (req. 2.3.9)
10. A TA who is a student(student_id referred to student_id in STUDENT relation) can assist in any course if it is open. Hence course_id, year, season from TA relation refers to IS_OPEN's respective fields. (req. 2.3.5 and req. 2.3.7)
11. A professor can instruct a course if it is open in that semester. Hence course_id, year, season from PROFESSOR relation refers to IS_OPEN's respective fields. (req. 2.3.5)

12. Each course may have pre-requisite courses. This is logged in PREREQUISITE relation. (req. 2.3.8)
13. We have ENROLLS relation in which studend_id refers to REGISTER(student_id) and course_id refers to IS_OPEN(course_id) and year, season refers to both REGISTER and IS_OPEN's year and season. This design ensures that student has registered in the semester and the course is offered in that semester. (req. 2.4.1)
14. Before we register a student, we need to check if the student has cleared prerequisites for that course. For this we have a STUDENT_PREREQUISITE(student_id, course_id, prereq_id, cleared) relation. Any entry in this table ensures that the student has cleared that respective course. (req. 2.4.1) This is referred by ENROLL relation.
15. FEEDBACK relation logs all the feedback for instructors given by students enrolled in that course. This ensures students to post feedback for the instructor of the course only if they enrolled. (req. 2.4.3)
16. ATTENDS relation has students enrolled in courses with their exam details for that course. This ensures, students only who take that course have grades on those exams.(req. 2.4.4)
17. STUDENT_EXAM tracks all the problems and scores received by the students who attend that exam. (req. 2.4.5)

Assumption in the schema:

1. The newly registered students are assumed to have passed their prerequisite courses. Hence, STUDENT_PREREQ should have an entry for a newly registered student before he/she enrolls for a course.

Advantages of the schema:

The database schema that we have designed is well defined and structured. Thus, this schema supports all the functions of TinyHub.

1. A student can enroll for a course only if he has passed all the prerequisite courses for that course. This can be achieved - when there is an entry in STUDENT_PREREQ relation, it is considered as that student has cleared prerequisites. In other words, a student can enroll only if his/her ID is present in the STUDENT_PREREQ table.
2. A student can only have grades A/B/C/D/F in each course. Our design makes this sure while inserting any new record in STUDENT_EXAM and ENROLL relations.

Disadvantages of the schema:

1. We didn't have a direct relationship between the **PROGRAM** entity and **COURSE** entity. The requirement didn't mention any relation between these two entities. Hence, if we need to query the courses which might belong to one particular program, we would not be able to perform this function.