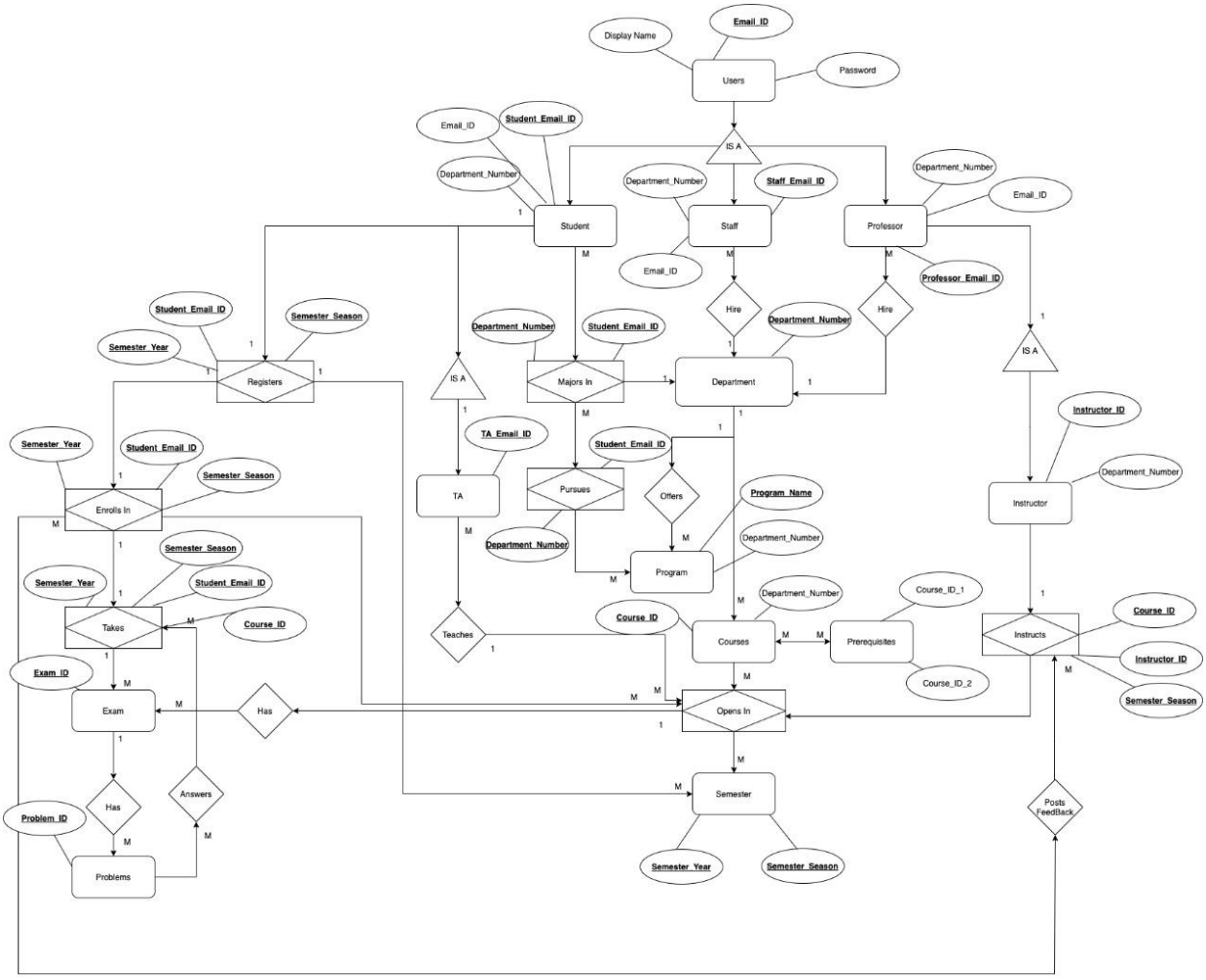
CSE 560 Project1 Report

E/R schema diagram:



https://buffalo.box.com/s/48r55zbedr38gsmxbwi7demcro3sekhl

Introduce your E/R schema:

User management

User is an entity is described through the set of attributes, user ID, email address, account and display name. This is because the users sign up their accounts with their email addresses and users need to set their passwords. Also, the users are allowed to change their display name. Furthermore, since one email address can be used to register only one account this will be used as a primary key. Furthermore, the user was defined as an entity using an ISA because it consists of three types of subclasses; the Students, Professors, and Staff. These are one-to-one relationship as one user can only be a student, or staff or a professor.

Department management

Department is an entity and has an identifier department number/Dept.ID. Every professor is hired by one department, and it's a many-to-one relationship as 1 professor are hired by 1 department but 1 department has many professors. Every staff is hired by one department, and it's a many-to-one relationship as 1 staff are hired by 1 department but 1 department has many staff. Students are allowed to major in different departments, thus it's a many-to-many relationship as 1 student can major in many departments and 1 department can have many students. Departments offer different programs, which a program can only belong to one department, thus it's a one-to-many relationship. Because 1 department offers many programs but 1 program belong to 1 department. Students can pursue different programs, but they must major in the department which is offering that program. This means the major-in relationship is an aggregate entity because the student must major in department, before they can pursue a program offered by the department. Thus, the pursue relationship is defined by the program entity and the major-in aggregate entity, which is a many-to many relationship, as 1 student can pursue many programs and 1 program can have many students.

Course management

Courses is an entity and has a course ID and a course title. Each department offers different courses, thus this is a 1-to-many relationship, as 1 department offers different courses but 1 courses belong to 1 department. Semester is an entity and is identified by a year and a season. A student may register in different semesters, this is a many-to- many relationship, as 1 student can register in many semesters and 1 semester can have many students and a semester can have many students. A course can be opened in different semesters which is a many-to-many relationship, as 1 course is open in many semesters and many courses are open in 1 semester. A course does not have to be opened every semester, means the year and season attribute of the semester will be empty or null. A course may contain prerequisite courses, this is a many-to-many relationship, as 1 prerequisite has many courses and a course can have many prerequisites.

An instructor must be a professor, is an ISA hierarchy as the professor is entity and the instructor is the subclass. This is a one-one relationship as 1 professor can only be 1 instructor and every instructor is also considered a professor. A TA must be a student, this is an ISA hierarchy as the student is entity and the TA is the subclass, as every TA is considered to be a student. This is a one-one relationship as 1 student can only be 1 TA.

Every course has TAs, and one instructor. Therefore, TA teaches a course is a many–to-1 because 1 TA can teach 1 course but many TA's belong to 1 course. The "teaches" relationship is defined by TA and "open" aggregate entity, because the course has to be open for the TA to teach the course. Instructor teaches a course is a many–to-1 because 1 instructor can teach many courses but 1 instructor teaches to only 1 course. The "instructs" relationship is defined by instructor and the open aggregate entity, because the course has to be open for the instructor can instruct the course. However, it is possible to change TAs or the instructor even during the semester, this means that the referenced tuple can be update. This will update the TA entity and the instructor entity and the relationships/aggregate entities connected to them.

Student-Course management

A student can enroll different courses. But that student can enroll in a course only if the student has registered in the semester the course is offered in that semester. This means that “enroll” relationship is defined by the “registers” aggregate entity and “open” aggregate entity. As a student needs to register to a semester and course has to be open in a semester before enrolling any course. This is a many-to-many relationship as 1 registered student can enroll in many open courses in a semester and 1 course can have many students.

Also, the student can enroll a course only if the student passes all the prerequisite courses, this is determined by the grade attribute which is a property of the “enroll” aggregate entity.

Furthermore, the student can enroll in a course only if It is being offered by a department, they are majoring in. Thus the “offer” relationship is defined by the department entity and the course entity, which is a 1-to-many relationship, as 1 department can offer many courses but 1 course belongs only to one department.

As well, the student can enroll a course only if the capacity of the course is not full. Thus, capacity is an attribute for open aggregate entity, since the course has to be open for us to know the capacity of the course.

Students will have a grade (F/D/C/B/A) with the course, which will be given after the student finishes the course. The grade is an attribute of the “enroll” aggregate entity, as when the student enrolls in the open course, they will get the grade for the course.

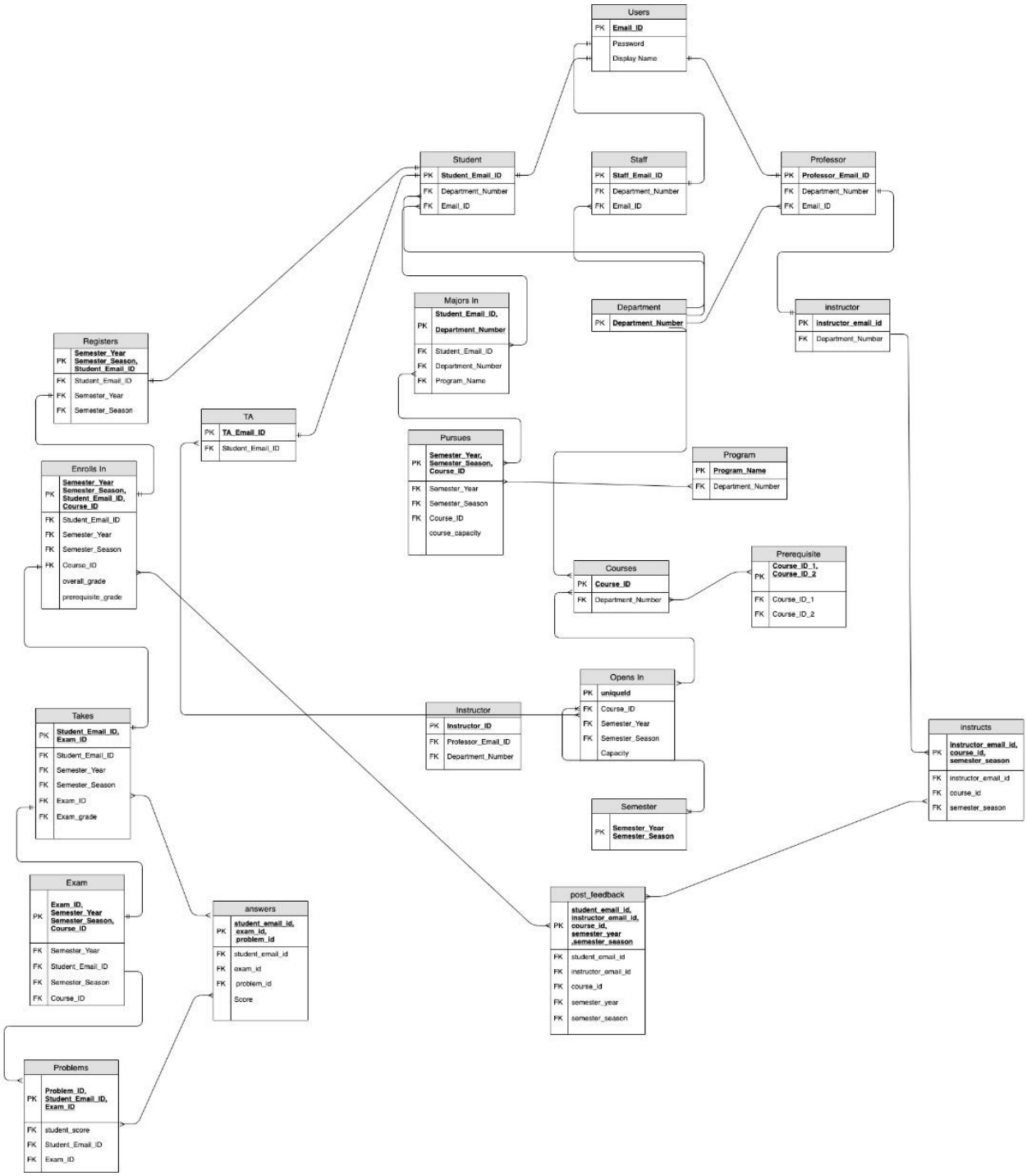
Students can post feedback for the instructor of the course in which they enrolled. Thus the “post feedback” relationship is defined by the “enroll” aggregate entity and the “instructs” aggregate entity. As the instructor must instruct an open course before students who are enrolled can post feedback. This is an many-to many relationship, as many student who are enrolled can post feedback for 1 instructor, and 1 instructor gets feedback from many students.

Each course has one or more exams. The course has to be open in a semester before the course can have an exam, thus the “have” relationship is defined by the “open” aggregate entity and the exam entity. This is a 1-to-many relationship, as 1 course can have many exams, but 1 exam belongs to 1 course. The students who take that course, have to be enrolled in the course before to take the exam. Thus “takes” relationship is defined by the “enroll” aggregate entity and the exam entity and the grade for those exams is an attribute for the “take” relationship.

Each exam has a number of problems. Thus, the problem is an entity as there are different types of problems. This is a 1-to-many-relationship, as 1 exam can have many problems but 1 problem can be fore 1 exam. Students have scores on those problems; thus, the enrolled student of an open course must take the exam before he can answer the problems in the exam. Thus, the “take“ is an aggregate entity and the “answers” relationship is defined by both the “take” aggregate entity and the problems entity. The scores are thus an attribute of the “answers” relationship.

Relational database schema:

Discuss briefly how you map the E/R schema to your relational database schema. If any design choice is made in the mapping process, illustrate and explain it briefly.



https://buffalo.box.com/s/aah74c26w2qsvx5wzq5dnad6p73fnhsj

Relational database schema justification:

Discuss briefly how your relational database schema satisfies all the requirements listed in section

User management

- We used a user as an ISA relationship because it consists of three types of subclasses; the Students, Professors, and Staff.
- We made email address as primary key as only one email address can be used for one account.
- We made account, username, password, display names as varchar.
- The Display name here is a varchar and is optional (can be NULL). This was achieved by adding a constraint unique for the display name.

Department management

- We satisfied the fact that every professor is hired by one department by making it a many-to-one relationship as 1 professor are hired by 1 department but 1 department has many professors.
- We satisfied every staff is hired by one department by making it a many-to-one relationship as 1 staff are hired by 1 department but 1 department has many staff.
- We satisfied that the students are allowed to major in different departments by having it a many-to-many relationship as 1 student can major in many departments and 1 department can have many students.
- To make departments offer different programs, we used a one-to-many relationship, since 1 department offers many programs, but 1 program belong to 1 department.
- The condition for student to pursue any program with a condition that he/she should major in the department which is offering the course, was achieved by aggregating the major-in relation to include student_id and department_number. This aggregate is then connected to the pursue relation consisting of program name. The program table consists of department code making a many-one relationship. This cyclic relationship between student, department and program using their primary keys helps asserting the condition in all many-many relation to enforce that the student majors in the same department which is offering the couse.

Course management

- We satisfied the requirement that each department offers different courses, by making the offer relation between department and courses a 1-to-many relationship, as 1 department offers different courses but 1 course belong to 1 department.
- We made semester entity has the combination of semester year and season in which is offered as the composite primary key.
- We made the ISA relationship between TA and the student as a TA is a subclass of student to achieve that requirement TA must be a student. This table contains the ta_email and the id of the course which the TA is teaching making a many-one relationship.
- We made the ISA relationship between professor and the instructor as instructor is a subclass of professor to achieve requirement instructor must be a professor.

- For the requirement that it is possible to change TAs or the instructor even during the semester, we allowed the referenced tuple can be updated
- We made register relation, between student and semester and made it a many-to- many relationship, as 1 student can register in many semesters and 1 semester can have many students and a semester can have many students, to satisfy the student may register in different semesters
- The perquisite for the course which is a many-many relationship was achieved by adding a self-containing many-many relation called “prerequisite” with the course. The condition for the pass/fail is enforced in student-course management system.
- The opens_in relation was created between course and semester, this also has the attribute called course capacity. This aggregate relation has the foreign keys course-id and semester, and their combination as primary key. Thus ensuring entries only when both the fields are present.
- To achieve the requirement every course has TAs, and one instructor, we made the “teaches” relation between TA and “open” aggregate entity, because the course has to be open for the TA to teach the course. We made the instruct relation between the instructor and the open aggregate entity, because the course has to be open for the instructor can instruct the course.

Student-Course management

- For the requirement a student can enroll different courses. We made the “enroll” relation between the “registers” aggregate entity and “open” aggregate entity. As a student needs to register to a semester and course has to be open in a semester before enrolling any course.
- We assigned the grade attribute to enroll aggregate entity to satisfy that the student can enroll a course only if the student passes all the prerequisite courses.
- We satisfied the requirement that the student can enroll in a course only if It is being offered by a department, they are majoring in. By making the “offer” relation between the department entity and the course entity.
- We achieved the requirement that student can enroll a course only if the capacity of the course is not full, by assigning capacity attribute to open aggregate entity.
- We assigned the grade attribute to the “enroll” relation, as when the student enrolls in the open course, they will get the grade for the course to satisfy the requirement that students will have a grade (F/D/C/B/A) with the course,
- We mad the “post feedback” relation between “enroll” aggregate entity and the “instructs” aggregate entity. As the instructor must instruct an open course before students who are enrolled can post feedback to achieve the requirement that students can post feedback for the instructor of the course in which they enrolled.
- We made “have” relation defined by the “open” aggregate entity and the exam entity, as the course has to be open in a semester before the course can have an exam. We made it a 1-to-many relationship, as 1 course can have many exams, but 1 exam belongs to 1 course to satisfy the requirement that each course has one or more exams.
- To achieve the requirement that each exam has several problems. We made the relation “has” a 1-to-many-relationship, as 1 exam can have many problems but 1 problem can be fore 1 exam.
- We made the “answers” relation between the “take” aggregate entity and the problems entity. As the student must take the exam before they can answer the problems in the exam.

- We assigned the scores attribute to the “answers” relation to enable us to find the scores on those problems for each student.

Further discussion:

Advantages of design:

- We try to reduce redundancies of having duplicated data in the relational schema.
- We created the tables with strong interlinking and constraints, so that all the fields are updated without additional commands.
- We used the primary keys as is, without needing to create artificial keys. This helps us to reduce space and enforce uniqueness while facilitating the queries.
- We reduced the number of relations in our design by joining to find the information about a user e.g. student’s grade in a course
- We made the update for changing the TA or instructor in a semester by using the null value.
- We handled a hierarchy set of entity sets with a single relation using the ISA hierarchies to save a lot of space e.g. TA isa student, instructor isa professor, and we used the single relation called user for staff, professor, student .

Disadvantages of design:

- We needed to have the condition a student must be registered in a semester before they can enroll in an open course, thus this constrain does not allow the student to enroll in an open course. But that’s part of the design requirement
- In order to enforce the course capacity, we need to add event triggers and procedures to run, with a counter mechanism which checks the condition every time there is an insert command. This implementation must be done during the functional implementation. A sample trigger is commented at the end of the sql file for the reference.