CSE 4/560 Project1: TinyHub

Chieh Ju Lu 50341634
Data Science
University at Buffalo
New York, United States
chiehjul@buffalo.edu

Mingmin Gong 50344720
Data Science
University at Buffalo
New York, United States
mgong3@buffalo.edu

Shreya Mathur 50340876
Data Science
University at Buffalo
New York, United States
shreyama@buffalo.edu

I. E/R schema

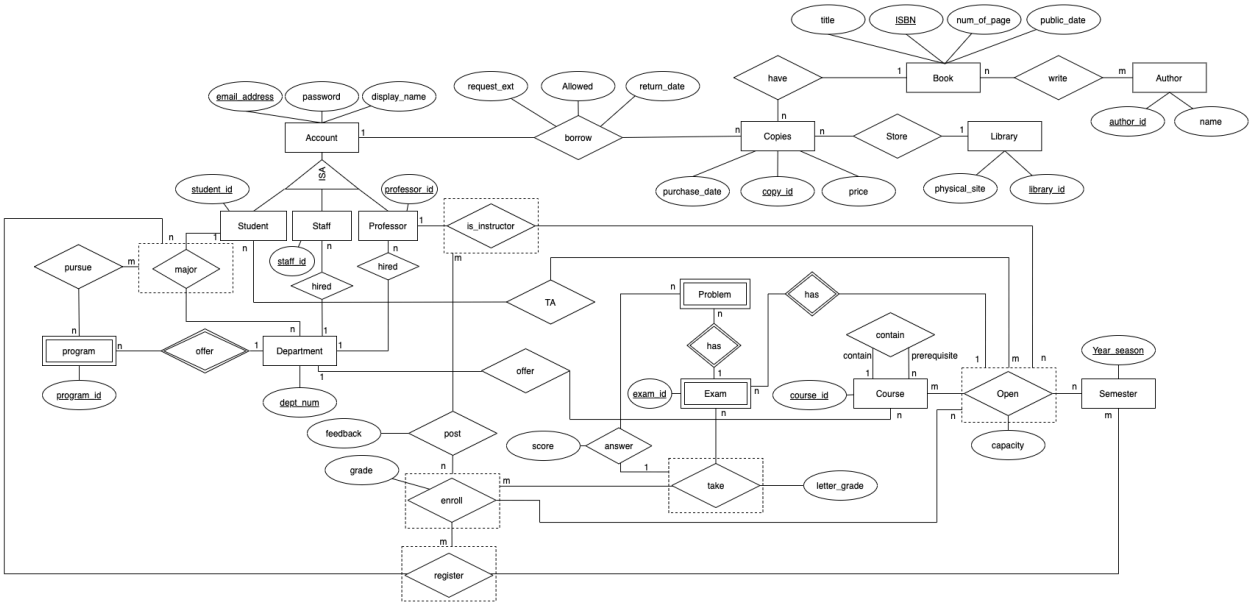

Fig 1. E/R schema

The plot above is the E/R diagram to present database schema for TinyHub, we use weak entity, aggregate and ISA hierarchies to design our database.

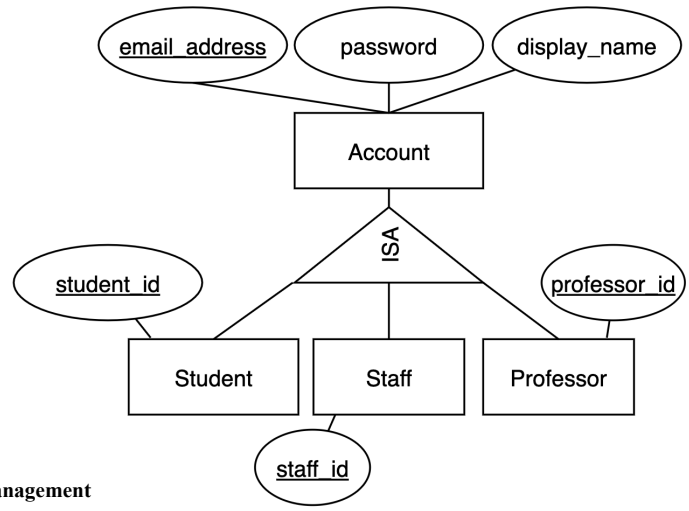

Fig 2. E/R Diagram for User Management

The following is the way we try to satisfy the requirements of User Management

1. Users sign up their accounts with their email addresses, and we can use email address to identify a unique account.
   - Create a entity “Account” with the key attribute “email_address”
2. Account can be of three types: Students, Professors, or Staffs.
   - Use ISA hierarchies to fulfill it, so then all the instances in Student, Stuff and Professor entities are also in Account entity.
3. Accounts need to set their passwords.
   - Add “password” attribute in “Account” entity, and we will set this attribute to not null for meeting this condition.
4. One account has an optional unique display name. Users are allowed to change their display name as long as it is unique.
   - Add “display_name” attribute in “Account” entity, and we will set this attribute to unique for meeting this condition.
5. One email address can be used to register only one account.
   - We will implement this in the next step.
6. Email, password, and display names are strings.
   - We will implement this in the next step.

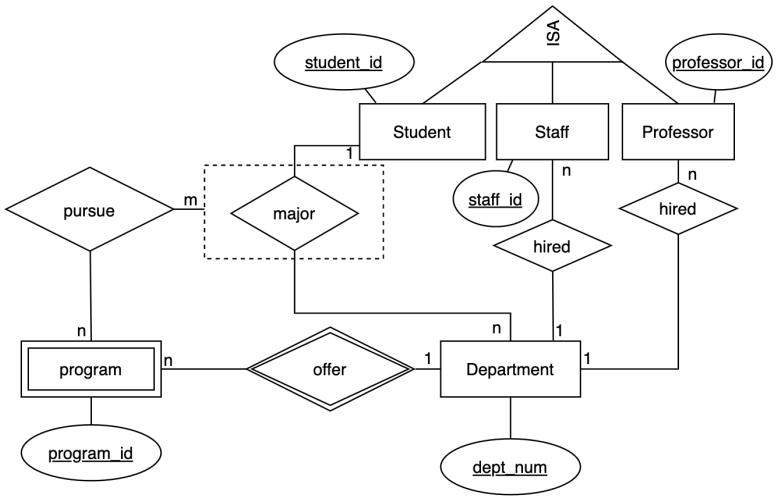

Fig 3. E/R Diagram for Department Management

The following is the way we try to satisfy the requirements of User Management

1. Each department has an identifier department number.
   - Create an entity “Department” with the primary key dept_num.
2. Every professor is hired by one department.
   - Create a many-to-one relationship “hired” between the entity “Professor” and “Department”.
3. Every staff is hired by one department.
   - Create a many-to-one relationship “hired” between the entity “Staff” and “Department”.
4. Departments offer different programs, which a program can only belong to one department.
   - It is a many-to-one relationship “offer” between entity “program” and “Department”, moreover, programs are dependent on the department, so we create a weak entity to fulfill this rule.
5. Students are allowed to major in different departments.
   - Create a one-to-many relationship “major” between the entity “Student” and “Department”.
6. Students can pursue different programs, but they must major in the department which is offering that program. e.g., Given a data science program offered by Computer science department, a student needs to major in computer science before pursue a data science program.
   - We create a many-to-many relationship “pursue” between the aggregation “major” and the entity “program”.

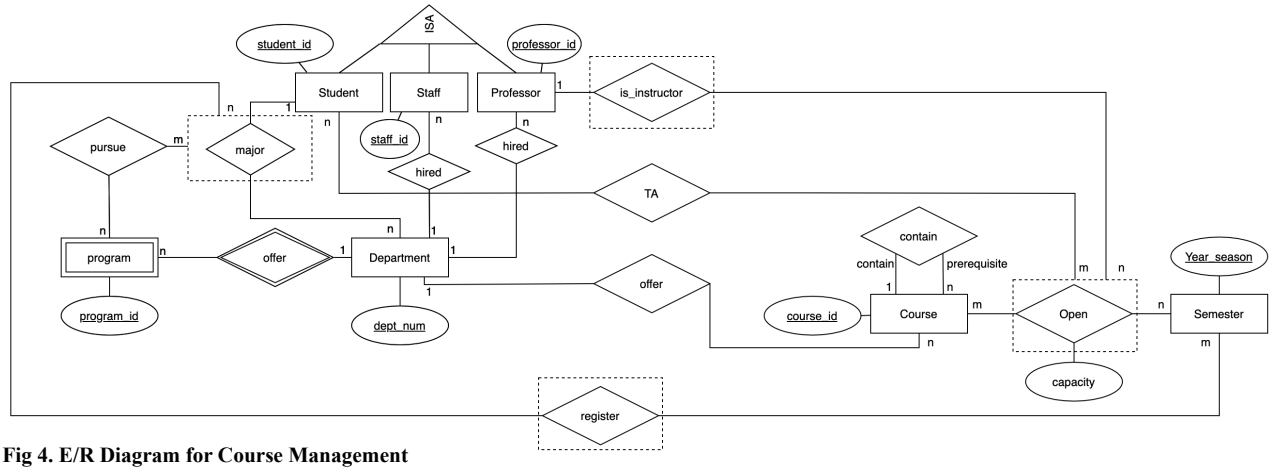

Fig 4. E/R Diagram for Course Management

The following is the way we try to satisfy the requirements of Course Management

1. Each department offers different courses.
   - Create a “Course” entity and a many-to-one relation “offer” between “Course” and “Department”.
2. A course can be opened in different semesters.
   - Create a “Semester” entity and a many-to-many relation “Open” between “Semester” and “Course”.
3. A semester is identified by a year and a season, e.g. 2020 Spring.
   - Add a Primary key “Year_season” of “Semester”
4. A course does NOT have to be opened every semester.
   - Many-to-many relation “Open” with “Course” and “Semester” is meeting the requirement when there is no relation established.
5. Every opened course has a capacity, which is the max student number in the class.
   - Add an attribute “capacity” in the entity “Open”
6. Every opened course has TAs, and one instructor. However, it is possible to change TAs or the instructor even during the semester.
   -Create aggregation of the relationship “Open” and build two relationships “TA” (many-to-many) and “is_instructor”(one-to-many), which allow the database to defined opened courses by relationship and entities. Moreover, the change of TAs and instructor will not affect other instances.
7. There might be multiple opened courses from the same course in a semester. e.g. CSE 101 has two sessions on 2020 Fall, which one of them has 100 as the capacity while the other session has 120.
   - Many-to-many relation “Open” with “Course” and “Semester” is meeting this requirement.
8. An instructor must be a professor.
   - The one-to-many relationship “is_instructor” is connected with the entity “Professor” and the aggregation of the relationship “Open”.
9. A TA must be a student.
   - The many-to-many relationship “TA” is connected with the entity “Student” and the aggregation of the relationship “Open”.
10. A course may contain prerequisite courses.
    - Create a one-to-many relationship “prerequisite” between the same entity set “Course”.
11. A student may register in different semesters.
    - Create a many-to-many relationship “register” between the aggregation “major” and the entity “Semester”.

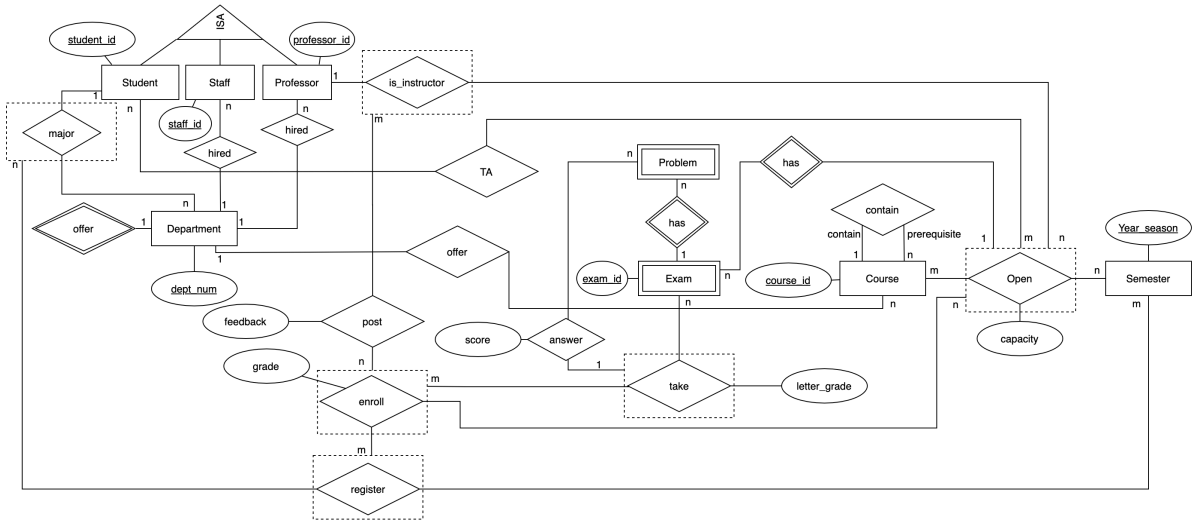

Fig 5. E/R Diagram for Student-Course Management

The following is the way we try to satisfy the requirements of Student-Course Management

1. A student can enroll different opened courses. But that student can enroll a course only if
   • That student has registered in the semester and the course is offered in that semester. i.e. A student need to register 2020 Spring before enrolling any course in 2020 Spring.
   - Create a many-to-many relationship “enroll” between two aggregate entity sets “register” and “Open”.

   • That student passes all the prerequisite courses
   - We will implement this in the next step.

   • It is being offered by a department they are majoring in
   - The aggregate entity set “register” contains the information of major.

   • The capacity of the course is not full
   - We will implement this in the next step.
2. Students will have a grade (F/D/C/B/A) with the course, which will be given after the student finishes the course.
   - Add an attribute “grade” in the relationship “enroll”.
3. Students can post feedbacks for the instructor of the course in which they enrolled.
   - Create a many-to-many relationship “post” between two aggregate entity sets “enroll” and “is_instructor”.
4. Each course has one or more exams. Students who take that course have letter grades on those exams.
   - We create a weak entity “Exam” that depends on the aggregation “Open” with a many-to-many relationship “has”. Also, create a many-to-many relationship “take” between the entity “Exam” and the aggregate entity set “enroll”, and add an attribute “letter_grade” in the “take” relationship.
5. Each exam has a number of problems. Students have scores on those problems.
   - Create a weak entity “Problem” with a many-to-one relationship to the entity “Exam”. Then, create a many-to-one relationship “answer” between the entity “problem” as well as the aggregate entity set “take” and add an attribute “score”.

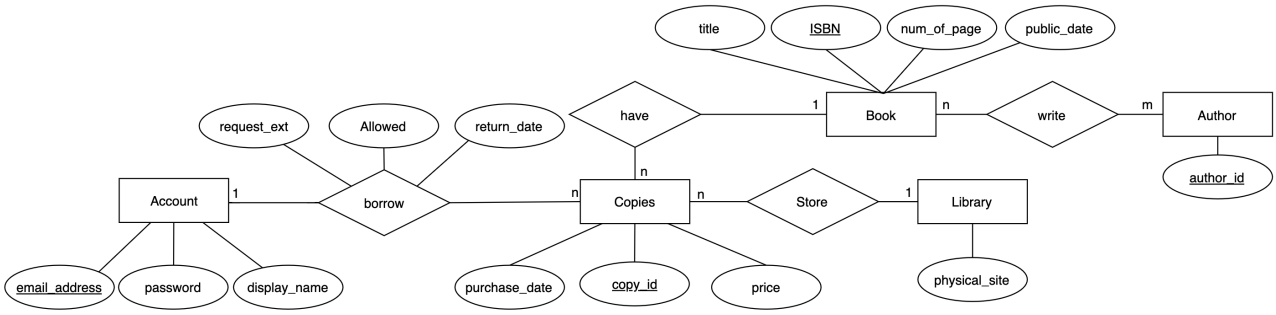

Fig 6. E/R Diagram for Library Management

The following is the way we try to satisfy the requirements of Library Management

1. Books have ISBN as the identifier. It also contains the title, author(s), number of pages, and publication date.
   - Create an entity “Book” with the primary key “ISBN”, and three attributes “title”, “num_of_page” and “public_date”.
2. An author may write multiple books. A book may also have more than one author.
   - Build a many-to-many relationship “write” between two entities “Book” and “Author”.
3. There may be more than one copy of a book in the library. e.g., There might be two copies of Animal Farm in the library.
   - Create an entities “copies” and use a many-to-one relationship “have” connected two entities “Copies” and “Book”. Then, create an entities “Library” and build a many-to-one relationship “Store” between the entity “Copies” and the entity “Library”.
4. There are different physical sites of the campus library.
   - Add an attribute “physical_site” in the entity “Library”.
5. The copies of the books may locate at different sites of the library.
   - The relationship “Store” is already satisfied this requirement.
6. Different copies of the same book may have different dates of purchase and prices.
   - Add attributes “purchase_date” and “price” in the entity “Copies”.
7. Any user can borrow books, and those books need to be returned within two weeks.
   - Create a one-to-many relationship “borrow” between two entities “Account” and “Copies”, then add an attribute “return_date” in the relationship “borrow”.
8. Users can request a one-week extension for each book they borrowed.
   - Add an attribute “request_ext” in the relationship “borrow”.
9. Users will not be allowed to borrow books if there is any book that has NOT been returned after the return date.
   - Add an attribute “allowed” in the relationship “borrow”.
10. Tinyhub prevents a user from returning a different copy of the same book. e.g., Given a user borrow one copy of Animal Farm, the user will not be allowed to return another copy of Animal Farm.
    - We already have enough information to satisfy this requirement, the relationship “borrow” will contain the primary key “copy_id”, which can help to identify different copy.

II. Relational database schema

1. Discuss briefly how you map the E/R schema to your relational database schema:

   First, we create table for each entity.
   Second, we make sure the attributes of each entity become the columns of the table with appropriate data type.
   Third, we set the table’s primary key and foreign key by figuring out all relationships(1 to 1, 1 to n, n to n, m to n) between entities.
   For example, the relationship between major, program, student and department. Since the major which offer program belongs to the department, and students who want to attend the program must enrolled the same major which offer the program. So we set the department number as the primary key on department table, the student id as the primary key and the department number as foreign key on student table, the major name as the primary key and the department number as foreign key on major table. For the program table, the program id is the primary key , the department number is foreign key, and the major name is the foreign key.

2. Discuss briefly how your relational database schema satisfies all the requirements:

   1) User management

(1) Users sign up their accounts with their email addresses, and we can use email address to identify a unique account:

Account (email_address)
PK(Account.email_address)

(2) Account can be of three types: Students, Professors, or Staffs:

Account (email_address, password, display_name)
PK(Account.email_address)

Student (student_id, sd_email_address)
PK(Student.student_id)

Professor (professor_id, pf_email_address)
PK(Professor.professor_id)

Staff (staff_id, sf_email_address)
PK(Staff.staff_id)

FK(Student.sd_email_address, Account.email_address)
FK(Professor.pf_email_address, Account.email_address)
FK(Staff.sf_email_address, Account.email_address)

(3) Accounts need to set their passwords.
- Set the attribute “password” not null.

(4) One account has an optional unique display name. Users are allowed to change their display name as long as it is unique.
- Set the attribute “display_name” unique.

(5) One email address can be used to register only one account:

Account (email_address, password, display_name)
PK(Account.email_address)

(6) Email, password, and display names are strings.
- Set the type of attribute “display_name”, “password”, “email_address” into varchar.

2) Department management

(1) Each department has an identifier department number:

Department (dept_num)
PK(Department.dept_num)

(2) Every professor is hired by one department:

Professor (professor_id, pf_email_address, hired_by)
PK(Professor.professor_id)

Department (dept_num)
PK(Department.dept_num)

FK(Professor.hired_by, Department.dept_num)

(3) Every staff is hired by one department:

Staff (staff_id, sf_email_address, hired_by)
PK(Staff.staff_id)

Department (dept_num)
PK(Department.dept_num)

FK(Staff.hired_by, Department.dept_num)

(4) Departments offer different programs, which a program can only belong to one department:

Department (dept_num)
PK(Department.dept_num)

program(program_id, belong_dept)
PK(program.program_id)

FK(program.belong_dept, Department.dept_num)

(5) Students are allowed to major in different departments:

Student (student_id, sd_email_address)
PK(Student.student_id)

Department (dept_num)
PK(Department.dept_num)

Major(major_id, stu_id, dep_id)
PK(Major.major_id)

FK(Major.stu_id, Student.Student_id)
FK(Major.dept_id, Department.dept_num)

(6) Students can pursue different programs, but they must major in the department which is offering that program. e.g., Given a data science program offered by Computer science department, a student needs to major in computer science before pursue a data science program :

Student (student_id, sd_email_address)
PK(Student.student_id)

Department (dept_num)
PK(Department.dept_num)

Major(major_id, stu_id, dep_id)
PK(Major.major_id)

program(program_id, belong_dept)
PK(program.program_id)

pursue(pursue_id, major_id, program_id)
PK(pursue.pursue_id)

FK(Major.stu_id, Student.Student_id)
FK(Major.dept_id, Department.dept_num)

FK(program.belong_dept, Department.dept_num)

FK(pursue.major_id, Major.major_id)
FK(pursue.program_id, program.program_id)

3) Course management

(1) Each department offers different courses:

Department (dept_num)
PK(Department.dept_num)

Course(course_id, dept_num)
PK(Course.course_id)

FK(Course.dept_num, Department.dept_num)

(2) A course can be opened in different semesters:

Course(course_id, dept_num)
PK(Course.course_id)

Semester(year_season)
PK(Semester.year_season)

Open(open_id, course_id, year_season)
PK(Open.open_id)

FK(Open.course_id, Course.course_id)
FK(Open.year_season, Semester.year_season)

(3) A semester is identified by a year and a season, e.g. 2020 Spring:

Semester(year_season)
PK(Semester.year_season)

(4) A course does NOT have to be opened every semester.
- The relation schema “Open” already fulfilled this requirement.

(5) Every opened course has a capacity, which is the max student number in the class:

Open(open_id, course_id, year_season, capacity)
PK(Open.open_id)

(6) Every opened course has TAs, and one instructor. However, it is possible to change TAs or the instructor even during the semester:

Open(open_id, course_id, year_season, capacity, instructor)
PK(Open.open_id)

TAs (ta_id, open_courseid)
PK(TAs.ta_id)

FK(TAs.open_courseid, Open.open_id)

(7) There might be multiple opened courses from the same course in a semester. e.g. CSE 101 has two sessions on 2020 Fall, which one of them has 100 as the capacity while the other session has 120.
- The relation schema Open(open_id, course_id, year_season, capacity) is already satisfied with this requirement, we can have multiple open courses in the same semester because the primary key is open_id.

(8) An instructor must be a professor:

Professor (professor_id, pf_email_address, hired_by)
PK(Professor.professor_id)

Open(open_id, course_id, year_season, capacity, instructor)
PK(Open.open_id)

FK(Open.instructor, Professor.professor_id)

(9) A TA must be a student:

Student (student_id, sd_email_address)
PK(Student.student_id)

Open(open_id, course_id, year_season, capacity, instructor)
PK(Open.open_id)

TAs (ta_id, open_courseid, student_id)
PK(TAs.ta_id)

FK(TAs.open_courseid, Open.open_id)
FK(TAs.student_id, Student.student_id)

(10) A course may contain prerequisite courses:

Course(course_id, dept_num)
PK(Course.course_id)

Prerequisite(pre_id, course_id)
PK(Prerequisite.pre_id, Prerequisite.course_id)

FK(Prerequisite.pre_id, Course.course_id)
FK(Prerequisite.course_id, Course.course_id)

(11) A student may register in different semesters:

Student (student_id, sd_email_address)
PK(Student.student_id)

Department (dept_num)
PK(Department.dept_num)

Semester(year_season)
PK(Semester.year_season)

register(register_id, student_id, year_season)
PK(register.register_id)

FK(register.student_id, Student.student_id)
FK(register.year_season, Semester.year_season)

4) Student-Course management

(1) A student can enroll different opened courses. But that student can enroll a course only if
• That student has registered in the semester and the course is offered in that semester. i.e. A student need to register 2020 Spring before enrolling any course in 2020 Spring:

Student (student_id, sd_email_address)
PK(Student.student_id)

Department (dept_num)
PK(Department.dept_num)

Semester(year_season)
PK(Semester.year_season)

register(register_id, student_id, year_season)
PK(register.register_id)

Open(open_id, course_id, year_season, capacity)
PK(Open.open_id)

Enroll(enroll_id, register_id, open_id)
PK(Enroll.enroll_id)

FK(register.student_id, Student.student_id)
FK(register.year_season, Semester.year_season)

FK(Enroll.register_id, register.register_id)
FK(Enroll.open_id, Open.open_id)

In the relation “Enroll”, we can get the students info (status of register and the semester) from mapping to the relation “register”, and the info of open course in that semester from mapping to the relation “Open”.

• That student passes all the courses
-In the relation “Enroll”, we can get the students info (enrolled course history) from mapping to the relation “register”, and the info of open course from mapping to the relation “Open” which can map to the relation “Course” to get the info of prerequisite course.

• It is being offered by a department they are majoring in
- We can get the major information through mapping the relation “Enroll” to the relation “register” which contains the major info.

• The capacity of the course is not full
- We can set the restriction of the opened courts enrollment from its capacity.

(2) Students will have a grade (F/D/C/B/A) with the course, which will be given after the student finishes the course:

register(register_id, student_id, year_season)
PK(register.register_id)

Open(open_id, course_id, year_season, capacity)
PK(Open.open_id)

Enroll(enroll_id, register_id, open_id, grade)
PK(Enroll.enroll_id)

FK(Enroll.register_id, register.register_id)

FK(Enroll.open_id, Open.open_id)

(3) Students can post feedbacks for the instructor of the course in which they enrolled.

Professor (professor_id, pf_email_address, hired_by)
PK(Professor.professor_id)

Open(open_id, course_id, year_season, capacity, instructor)
PK(Open.open_id)

FK(Open.instructor, Professor.professor_id)

Enroll(enroll_id, register_id, open_id, grade)
PK(Enroll.enroll_id)

post(post_id, enroll_id, open_id, feedback)
PK(post.post_id)

FK(post.enroll_id, Enroll.enroll_id)
FK(post.open_id, Open.open_id)

We can get the instructor info from the relation “open”, because each opened course can only have an instructor.

(4) Each course has one or more exams. Students who take that course have letter grades on those exams:

Open(open_id, course_id, year_season, capacity, instructor)
PK(Open.open_id)

Exam(exam_id, open_id)
PK(Exam.exam_id)

FK(Exam.open_id, Open.open_id)

Enroll(enroll_id, register_id, open_id, grade)
PK(Enroll.enroll_id)

take(take_exam_id, enroll_id, exam_id, letter_grade)
PK(take.take_exam_id)

FK(take.enroll_id, Enroll.enroll_id)
FK(take.exam_id, Exam.exam_id)

To get the information of students who in the course, we can mapping to the relation “Enroll” and also mapping to the course exams through the relation “Enroll” and “Exam”, then store the letter grades in the attribute “letter_grade” in the “take” relation.

(5) Each exam has a number of problems. Students have scores on those problems:

Exam(exam_id, open_id)
PK(Exam.exam_id)

problems(prob_id, exam_id)
PK(problems.prob_id)

FK(problems.exam_id, Exam.exam_id)

take(take_exam_id, enroll_id, exam_id, letter_grade)
PK(take.take_exam_id)

answer(ans_id, take_exam_id, prob_id, score)
PK(answer.ans_id)

FK(answer.take_exam_id, take.take_exam_id)
FK(answer.prob_id, problems.prob_id)

5) Library management The TinyHub is connected with the library system on campus. Here are the requirements.

(1) Books have ISBN as the identifier. It also contains the title, author(s), number of pages, and publication date:

Book(ISBN, title, num_of_page, public_date, authors)
PK(Book.ISBN)

(2) An author may write multiple books. A book may also have more than one author:

Book(ISBN, title, num_of_page, public_date)
PK(Book.ISBN)

Author(author_id, author_name)
PK(author_id)

write(write_id, ISBN, author_id)
PK(write.write_id)

FK(write.ISBN, Book.ISBN)
FK(write.author_id, Author.author_id)

(3) There may be more than one copy of a book in the library. e.g., There might be two copies of Animal Farm in the library:

Book(ISBN, title, num_of_page, public_date)
PK(Book.ISBN)

Library(library_id)
PK(Library.library_id)

Copies(copy_id, ISBN, library_id)
PK(Copies.copy_id)

FK(Copies.ISBN, Book.ISBN)
FK(Copies.library_id, Library.library_id)

(4) There are different physical sites of the campus library:

Library(library_id, physical_site)
PK(Library.library_id)

(5) The copies of the books may locate at different sites of the library.
- We can get information of sites through the relation Copies(copy_id, ISBN, library_id) and Library(library_id, physical_site).

(6) Different copies of the same book may have different dates of purchase and prices:

Copies(copy_id, ISBN, library_id, price, purchase_date)
PK(Copies.copy_id)

(7) Any user can borrow books, and those books need to be returned within two weeks:

Account (email_address, password, display_name)
PK(Account.email_address)

Copies(copy_id, ISBN, library_id, price, purchase_date)
PK(Copies.copy_id)

Borrow(borrow_id, copy_id, user_email, return_date)
PK(Borrow.borrow_id)

FK(Borrow.copy_id, Copies.copy_id)
FK(Borrow.user_email, Account.email_address)

(8) Users can request a one-week extension for each book they borrowed:

Borrow(borrow_id, copy_id, user_email, return_date, request_ext)
PK(Borrow.borrow_id)

(9) Users will not be allowed to borrow books if there is any book that has NOT been returned after the return date.

Account (email_address, password, display_name)
PK(Account.email_address)

Copies(copy_id, ISBN, library_id, price, purchase_date)
PK(Copies.copy_id)

Borrow(borrow_id, copy_id, user_email, return_date, request_ext, allowed)
PK(Borrow.borrow_id)

FK(Borrow.copy_id, Copies.copy_id)
FK(Borrow.user_email, Account.email_address)

We can get whole the needed information such as user and borrowed record, then decide the status of allowing the user to borrow books.

(10) Tinyhub prevents a user from returning a different copy of the same book. e.g., Given a user borrow one copy of Animal Farm, the user will not be allowed to return another copy of Animal Farm.
- We already have copy_id to identify every different copy (even copies of the same book), so the relation Borrow(borrow_id, copy_id, user_email, return_date, request_ext) fulfilled this requirement.

III. Further discussion

We use the serial number id as the primary key instead of the detailed information to prevent the redundancies of information in the different tables, which decreases the storage of the same data (ex. email) in ours DB. Also, we create several tables to store the information of relationship instead of flattening all the data in one table, which enables us to update the value easily. Hence, our design is convenient to maintain. However, due to the complexity of this DB schema, it is not intuitive and will need a high intensity of computation to get the needed information.