PROJECT 1

CSE560: TinyHub

Rasita Pai
UB Name: rasitash

Varsha Ravichandiran
UB Name: varshara

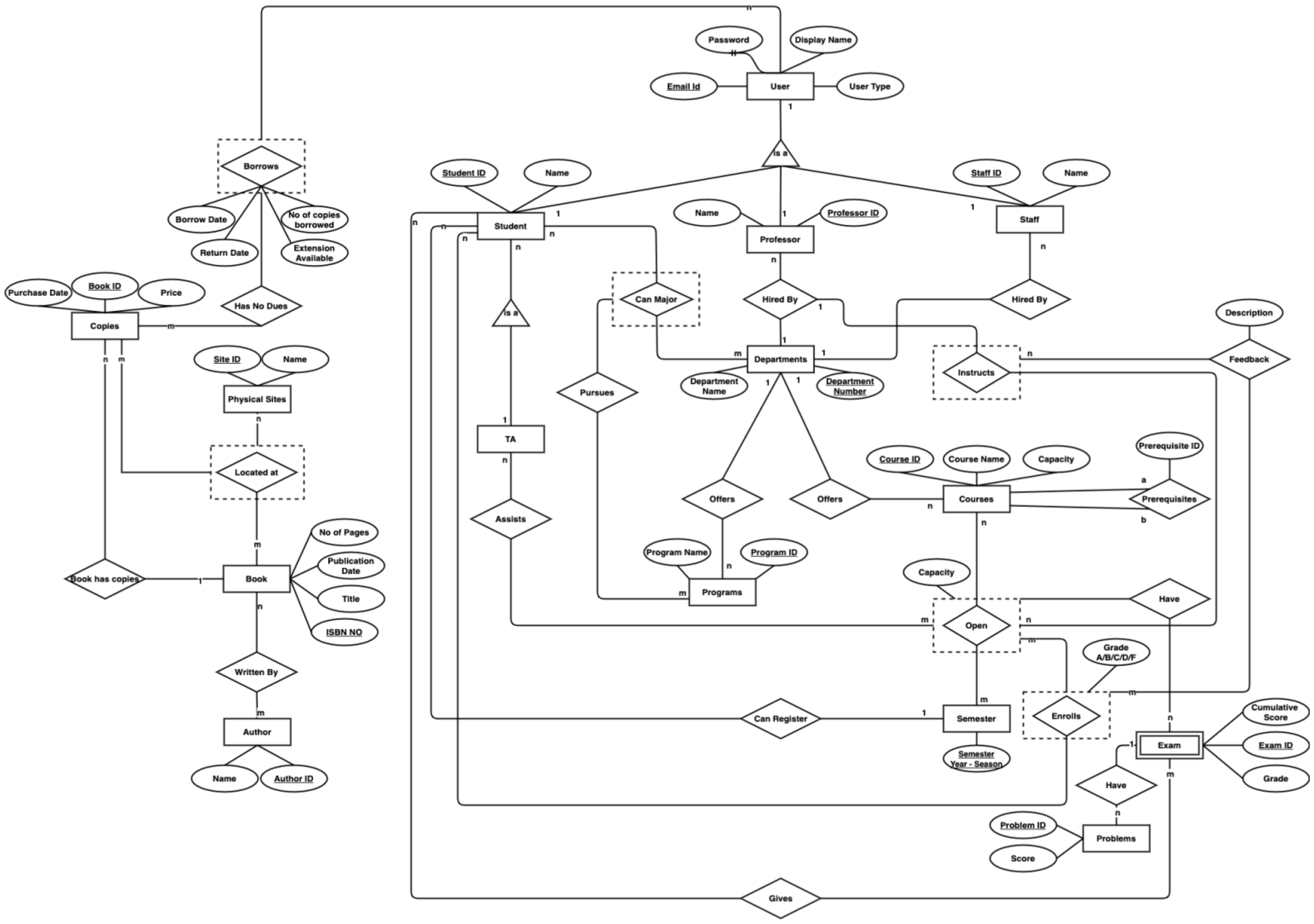
Sanidhya Chopde
UB Name: schopde

Objective:

The objective of this project is to design and implement database schema for TinyHub, which is a course enrollment system. It provides simple functions, main functions for Tiny-Hub are the following:

- User management
- Department management
- Course management
- Student-Course management
- Library management

E/R Schema:



User
Password
Display Name
Email Id
User Type
is a
Student
Student ID
Name
Professor
Professor ID
Staff
Staff ID
Departments
Department Name
Department Number
Hired By
Can Major
Pursues
Programs
Program Name
Program ID
Offers
Courses
Course ID
Course Name
Capacity
Prerequisites
Prerequisite ID
Instructs
Feedback
Description
TA
Assists
Open
Have
Semester
Semester Year - Season
Can Register
Enrolls
Grade A/B/C/D/F
Exam
Exam ID
Cumulative Score
Grade
Problems
Problem ID
Score
Gives
Borrows
Borrow Date
Return Date
No of copies borrowed
Extension Available
Has No Dues
Copies
Book ID
Purchase Date
Price
Physical Sites
Site ID
Located at
Book has copies
Book
No of Pages
Publication Date
Title
ISBN NO
Written By
Author
Author ID

Relational Database Schema:

The following tables are a part of the schema:

- USERS
- DEPARTMENT
- COURSES
- PROGRAMS
- SEMESTER
- STUDENT
- PROFESSOR
- STAFF
- MAJORS
- PROF_HIREDBY
- STAFF_HIREDBY
- PURSUES
- REGISTERS
- OPEN_COURSE
- PREREQUISITES
- ENROLLS
- TA
- ASSISTS
- INSTRUCTS
- FEEDBACK
- PROGRAMS_OFFERS
- COURSES_OFFERS
- EXAM
- PROBLEMS
- HAVE_EXAMS
- HAVE_PROBLEMS
- GIVES_EXAMS
- BORROWS_FROM
- PHYSICAL_SITES
- BOOK
- AUTHOR
- COPIES
- HAS_NO_DUES
- LOCATED_AT
- BOOK_HAS_COPIES
- WRITTEN_BY

Schema:

1. USERS (emailId, displayName, password, userType)
   Primary Key: USERS.emailId
   Unique Key: USERS.displayName

2. DEPARTMENT (departmentNo, departmentName)
   Primary Key: DEPARTMENT.departmentNo

3. COURSES (courseId, courseName, capacity)
   Primary Key: COURSES.courseId
   Foreign Key: DEPARTMENT.departmentNo

4. PROGRAMS (programId, programName)
   Primary Key: PROGRAMS.programId
   Foreign Key: DEPARTMENT.departmentNo

5. SEMESTER (year_season)
   Primary Key: SEMESTER.year_season

6. STUDENT (studentId, studentName)
   Primary Key: STUDENT.studentId
   Foreign Key: USERS.emailId

7. PROFESSOR (professorId, professorName)
   Primary Key: PROFESSOR.professorId
   Foreign Key: USERS.emailId
   Foreign Key: DEPARTMENT.departmentNo

8. STAFF (staffId, staffName)
   Primary Key: STAFF.staffId
   Foreign Key: USERS.emailId
   Foreign Key: DEPARTMENT.departmentNo

9. MAJORS
   Foreign Key: STUDENT.studentId
   Foreign Key: DEPARTMENT.departmentNo

10. PROF_HIREDBY
    Foreign Key: PROFESSOR.professorId
    Foreign Key: DEPARTMENT.departmentNo

11. STAFF_HIREDBY
    Foreign Key: STAFF.staffId
    Foreign Key: DEPARTMENT.departmentNo

12. PURSUES
    Foreign Key: MAJORS.studentId
    Foreign Key: PROGRAMS.departmentNo
    Foreign Key: PROGRAMS.programId

13. REGISTERS
    Foreign Key: MAJORS.studentId
    Foreign Key: SEMESTER.year_season

14. OPEN_COURSE (capacity)
    Primary Key: OPEN_COURSE.courseId, OPEN_COURSE.year_season
    Foreign Key: COURSES.courseId
    Foreign Key: SEMESTER.year_season
    Foreign Key: PROFESSOR.professorId

15. PREREQUISITES (prerequisite_id)
    Foreign Key: COURSES.courseId
    Foreign Key: (PREREQUISITES.prerequisiteId, COURSES.courseId)

16. ENROLLS (grade)
    Primary Key: ENROLLS.studentId, ENROLLS.courseId, ENROLLS.year_season
    Foreign Key: STUDENT.studentId
    Foreign Key: OPEN_COURSE.year_season
    Foreign Key: OPEN_COURSE.courseId

17. TA
    Foreign Key: STUDENT.studentId

18. ASSISTS
    Foreign Key: TA.studentId
    Foreign Key: OPEN_COURSE.year_season
    Foreign Key: OPEN_COURSE.courseId

19. INSTRUCTS
    Foreign Key: OPEN_COURSE.courseId
    Foreign Key: OPEN_COURSE.year_season
    Foreign Key: PROF_HIREDBY.professorId

20. FEEDBACK (description)
    Foreign Key: ENROLLS.studentId
    Foreign Key: INSTRUCTS.professorId
    Foreign Key: ENROLLS.courseId

21. PROGRAMS_OFFERS
    Foreign Key: PROGRAMS.departmentNo
    Foreign Key: PROGRAMS.programId

22. COURSES_OFFERS
    Foreign Key: COURSES.departmentNo
    Foreign Key: COURSES.courseId

23. EXAM (examId, cumulativeScore, grade)
    Primary Key: EXAM.examId
    Foreign Key: COURSES.courseId

24. PROBLEMS (problemId, score)
    Primary Key: PROBLEMS.problemId
    Foreign Key: EXAM.examId

25. HAVE_EXAMS
    Foreign Key: EXAM.examId
    Foreign Key: OPEN_COURSE.courseId
    Foreign Key: OPEN_COURSE.year_season

26. HAVE_PROBLEMS
    Foreign Key: PROBLEMS.examId
    Foreign Key: PROBLEMS.problemId

27. GIVES_EXAMS
    Foreign Key: EXAM.examId
    Foreign Key: STUDENT.studentId

28. BORROWS_FROM (borrowDate, returnDate, copiesBorrowed, extensionAvailable)
    Foreign Key: USERS.emailId

29. PHYSICAL_SITES (siteId, siteName)
    Primary Key: PHYSICAL_SITES.siteId

30. BOOK (ISBN_No, title, nofPages, publicationDate)
    Primary Key: BOOK.ISBN_No

31. AUTHOR (authorId, authorName)
    Primary Key: AUTHOR.authorId

32. COPIES(bookId, purchaseDate, price)
    Primary Key: COPIES.bookId

33. HAS_NO_DUES
    Foreign Key: BORROWS.emailId
    Foreign Key: COPIES.bookId

34. LOCATED_AT
    Foreign Key: BOOK.ISBN_No
    Foreign Key: PHYSICAL_SITES.siteId
    Foreign Key: COPIES.bookId

35. BOOK_HAS_COPIES
    Foreign Key: BOOK.ISBN_No
    Foreign Key: COPIES.bookId

36. WRITTEN_BY
    Foreign Key: BOOK.ISBN_No
    Foreign Key: AUTHOR.authorId

The main functions provided by TinyHub include:

User management:

- Users sign up their accounts with their email addresses, so we have used the email address attribute to identify a unique user account.
- The entity USER has attributes as email_id, display_name, password, user_type.
- Each user will have a mandatory email address associated with it and an optional display name which is unique for each user.
- Since user accounts can be further classified into 3 types, namely student, professor or staff, we have formed 3 separate entities for each type as STUDENT, PROFESSOR, STAFF and connected it to USER entity via a IS-A relationship.

- For the STUDENT entity, we uniquely identify a student using the student_id and the student's email_id references the email_id from USER entity. Similarly, for PROFESSOR entity, we have the professor_id as the key attribute and the professor's email_id references the user entity's attribute email_id. Similarly, for the STAFF entity, we identify a staff using the staff_id and this entity references the user entity as well.

Department management:

- We have DEPARTMENT as an entity, with attributes as department_number and department_name, where department_number is the key attribute.
- We create prof_hiredby relation between department and professor, depicting that each professor is hired by one department. Since there would be many professors in a department, the cardinality of this relation is one to many (1:N) where 1 department is mapped to N professors.
- Similarly, we create a staff_hiredby relation between staff and department, depicting that each staff is hired by one department. Since there can be many staff members in a department, this relation has a cardinality of one to many (1:N) where 1 department is mapped to N staff.
- Departments offer different programs, so we have an entity PROGRAMS which is connected to the DEPARTMENT entity through a OFFERS relation. Since a program can only belong to one department but one department can offer many programs, so the cardinality of this relation is one to many with 1 department mapped to N programs.
- Students are allowed to major in different departments. The STUDENT entity is connected to the DEPARTMENT entity via a CAN_MAJOR relationship whose cardinality is many-to-many (M:N) since a student can major in multiple departments and a department can have many students majoring in it.
- Students can pursue different programs, but they must major in the department which is offering that program. So we have a PURSUES relationship that connects the STUDENT entity to the PROGRAM entity through the CAN_MAJOR aggregated relationship. Since there can be many students pursuing a particular program and a certain student can pursue multiple programs, we have the cardinality of this relationship as many to many (M:N).

Course management:

- We have COURSES as an entity, with attributes as course_id, course_name and capacity, where course_id is the key attribute. Capacity attribute contains the capacity of the students in that course.
- We create OFFERS relation between departments and courses, depicting that each department offers multiple courses. Therefore, the cardinality of this relation is one to many (1:N) where 1 department is mapped to N courses.
- A course may contain prerequisite courses hence a prerequisite recursive relation is created to courses to get all the prerequisite courses for that particular course. Prerequisite relation has a prerequisite_id which uniquely identifies a course.
- We have SEMESTER as an entity, with attributes as year_season where year_season is the key attribute.

- Courses can be opened in multiple semesters, so the entity semester is connected to the courses through an OPEN_COURSE relation. Since each semester has multiple courses and each course can be opened in multiple semesters, the cardinality of this relation is many to many (N:M). Open_course relation has a capacity attribute that contains the capacity of students in that semester.
- Any course can be instructed by any professor and hence a relation INSTRUCTS is created and the cardinality is one to one(1:1). Any student can be TA for multiple courses which are opened in that semester.
- A student can register to the semesters through a relation REGISTERS and since each student can get registered to one semester at a time and each semester can be registered by multiple students the cardinality is one to many (N:1).

Student – Course Management:

- A student can enroll in a course only if that student has registered for a course which is opened. This can be achieved by creating a relation called ENROLLS between STUDENT and OPEN.
- The capacity attribute in the OPEN entity allows a student to enroll in a course only if the capacity of the course is not full.
- The grade attribute in the entity ENROLLS is used for storing the grade that a student obtains for an enrolled course.
- The relation FEEDBACK between ENROLLS and INSTRUCTS allows for the students to post feedback for the instructor of the enrolled course.
- The EXAM entity is created with attributes exam_id, cumulative_score, and grade as each open course will have multiple exams.
- The entity PROBLEMS with the attributes problem_id and score is created for the requirement each Exam has many problems and each problem has the score.

Library management:

- Users borrow books from the library. Library has many physical sites that contain many copies of multiple books. Hence COPIES is an entity that has attributes book_id, purchase_date and price where book_id is the primary key. Also BOOK entity has attributes like ISBN_No, title and no_of_pages, publicationDate where ISBN_No is the key attribute and PHYSICAL_SITES entity has site_id and site_name where site_id is the key attribute.
- The user burrows copies of books using BORROWS relation. Since each user can borrow multiple copies of books and each book is borrowed by multiple users the cardinality is many to many (N:M). Borrows relation has details like borrow_date, return_date, no_of_copies_borrowed and extension_availabe.
- Book is written by authors hence an AUTHOR entity with author_id and author_name is created with author_id as the key attribute. Since a book can be written by multiple authors and an author can write multiple books the cardinality relation is (N:M). A WRITTEN_BY relation is connected between book and author entities.

- A user can borrow a book only if the user has no dues. Hence HAS_NO_DUES relation exists between borrows and copies entities and the relation checks for the valid user and valid book.
- Copies of books are located at different physical sites and hence they are connected using LOCATED_AT relation. One copy of a book can be in multiple physical sites and one physical site can have multiple copies of books hence the cardinality relation is many to many(N:M).
- The entities book and copies are connected using BOOK_HAS_COPIES relation and their cardinality is one to many (1:N) since a book can have multiple copies.

Schema Justification

2.1

- 2.1.1 We have created USERS table with emailId as the primary key attribute which identifies a unique user.
- 2.1.2 We have generalized the users of TinyHub which are STUDENT, PROFESSOR and STAFF entities with 'IS A' relationship to USERS entity.
- 2.1.3 We have created a password attribute in the USERS entity which enables the users to set a password.
- 2.1.4 We have created a display_name attribute in the USERS entity as a unique key, which enables the TinyHub users to set a unique display name.
- 2.1.5 Requirement justified in 2.1.1.
- 2.1.6 We have created the attributes emailId, password and display_name of varchar datatype

2.2

- 2.2.1 We have created DEPARTMENT table with departmentNo as the primary key attribute which identifies a unique department.
- 2.2.2 To satisfy this requirement we have made a relation PROF_HIREDBY between DEPARTMENT and PROFESSOR with cardinality many to one.
- 2.2.3 To satisfy this requirement we have made a relation STAFF_HIREDBY between DEPARTMENT and STAFF with cardinality many to one.
- 2.2.4 To satisfy this requirement we have created an entity called PROGRAMS with a primary key programId and an attribute programName. We have made a relation PROGRAMS_OFFERS between DEPARTMENT and PROGRAMS with cardinality one to many.
- 2.2.5 To satisfy this requirement we have made an aggregate relationship MAJORS between STUDENT and DEPARTMENT with cardinality many to many.
- 2.2.6 To satisfy this requirement we have create a relation PURSUES which is connected between the aggregate relationship MAJORS and PROGRAMS with cardinality many to many.

2.3

- 2.3.1 We have made an entity COURSES with attributes courseId(primary key), courseName and capacity and connected it with DEPARTMENT entity by the COURSES_OFFERS relation with cardinality many to one.

2.3.2 We have created the entity SEMESTER with the attribute year_season as the primary key and connected it to COURSES via an aggregate relationship OPEN_COURSE with cardinality many to many.
2.3.3 Requirement justified in 2.3.2.
2.3.4 Requirement justified in 2.3.2.
2.3.5 We have created an attribute capacity in the OPEN_COURSE relation that specifies the max students who can enroll in that particular semester.
2.3.6 We have created the entity TA as 'IS A' relation to STUDENT and connected it to the OPEN_COURSE relation via the relation ASSISTS. We also have created a relationship INSTRUCTS between PROF_HIREDBY and OPEN_COURSE to justify the requirement.
2.3.7 We have the cardinality many to many between COURSES and SEMESTER via the relation OPEN_COURSE to justify this requirement.
2.3.8 This requirement is satisfied by the INSTRUCTS relation.
2.3.9 Requirement justified in 2.3.6.
2.3.10 We have created a recursive relationship called PREREQUISITES with COURSES with an attribute prerequisite_id and cardinality many to many to satisfy this requirement.
2.3.11 We have created the relation REGISTERS between STUDENT and SEMESTER with the cardinality many to 1 to satisfy this requirement.

2.4

2.4.1 We have created and aggregate relation ENROLLS between OPEN_COURSE and STUDENT with cardinality many to many. Rest of the requirement already covered in the section 2.3.
2.4.2 We have created a grade attribute with check constraint in ENROLLS to satisfy this requirement.
2.4.3 We have created a relation FEEDBACK with attribute descriptions between ENROLLS and INSTRUCTS with cardinality many to many to satisfy this requirement.
2.4.4 We have created an entity EXAM with examId (primary key), cumulative_score and grade attributes and joined it with OPEN_COURSE with the relation HAVE_EXAMS with cardinality many to many. Also, the EXAM entity is connected with the STUDENT entity by the relation GIVES_EXAM with cardinality many to many.
2.4.5 We have created the entity PROBLEMS with attributes problemId and score and connected it with EXAM by the relation HAVE_PROBLEMS with cardinality many to one.

2.5

2.5.1 We have created the entity BOOK with attributes ISBN_No (primary key), title and no_of_Pages to satisfy this requirement.
2.5.2 To satisfy this, we have created an entity called AUTHOR with attributes name and authorId and connected it with BOOK entity using the relation WRITTEN_BY with cardinality many to many.
2.5.3 To satisfy this, we have created an entity COPIES with attributes bookId (primary key), purchaseDate and price and joined it with BOOK with the relation BOOK_HAS_COPIES with cardinality many to one.

2.5.4 To satisfy this, we have created an entity PHYSICAL_SITES with attributes siteID (primary key) and siteName and connected it with COPIES and BOOK with the cardinality many to many.
2.5.5 Requirement justified in 2.5.4.
2.5.6 Requirement justified in 2.5.3.
2.5.7 To satisfy this, we have created an aggregate relation BORROWS with attributes borrowDate, returnDate, copiesBorrowed, extensionAvailable and also a check constraint chk_date to check if the book has been returned within 3weeks (2 weeks + 1week extension). We have created BORROWS between USERS and COPIES with a relation HAS_NO_DUES in between with cardinality many to many.
2.5.8 Requirement justified in 2.5.7.
2.5.9 Requirement justified in 2.5.7.
2.5.10 Requirement justified in 2.5.3.

Advantages:

- We have satisfied all the mentioned requirements for TinyHub as well as the Library Management System.
- Both our systems are well connected and structured using aggregate relations and corresponding entities
- We have provided proper check constraints for the grade attribute in TinyHub as well as the return date attribute in Library Management System.

Disadvantages:

- The requirement didn’t specify any direct connection between PROGRAMS and COURSES which made it ambiguous to distinguish which particular course is offered by which program.
- The requirement didn’t mention if the same copies of a book will have different ISBN NO’s or not and we proceeded with the assumption that ISBN numbers are same for all copies of a book. We later rectified this by adding a bookId attribute to the COPIES entity.