# CSE 560 - Project 01:
# TinyHub - a course enrollment website

Md Moniruzzaman Monir, # 50291708

October 3, 2019

## 1 E/R Schema

The ER schema of `TinyHub` is given below. The schema shows the various strong and weak entity types, the `isa` relationships, and how entity sets are related and cardinalities between them.

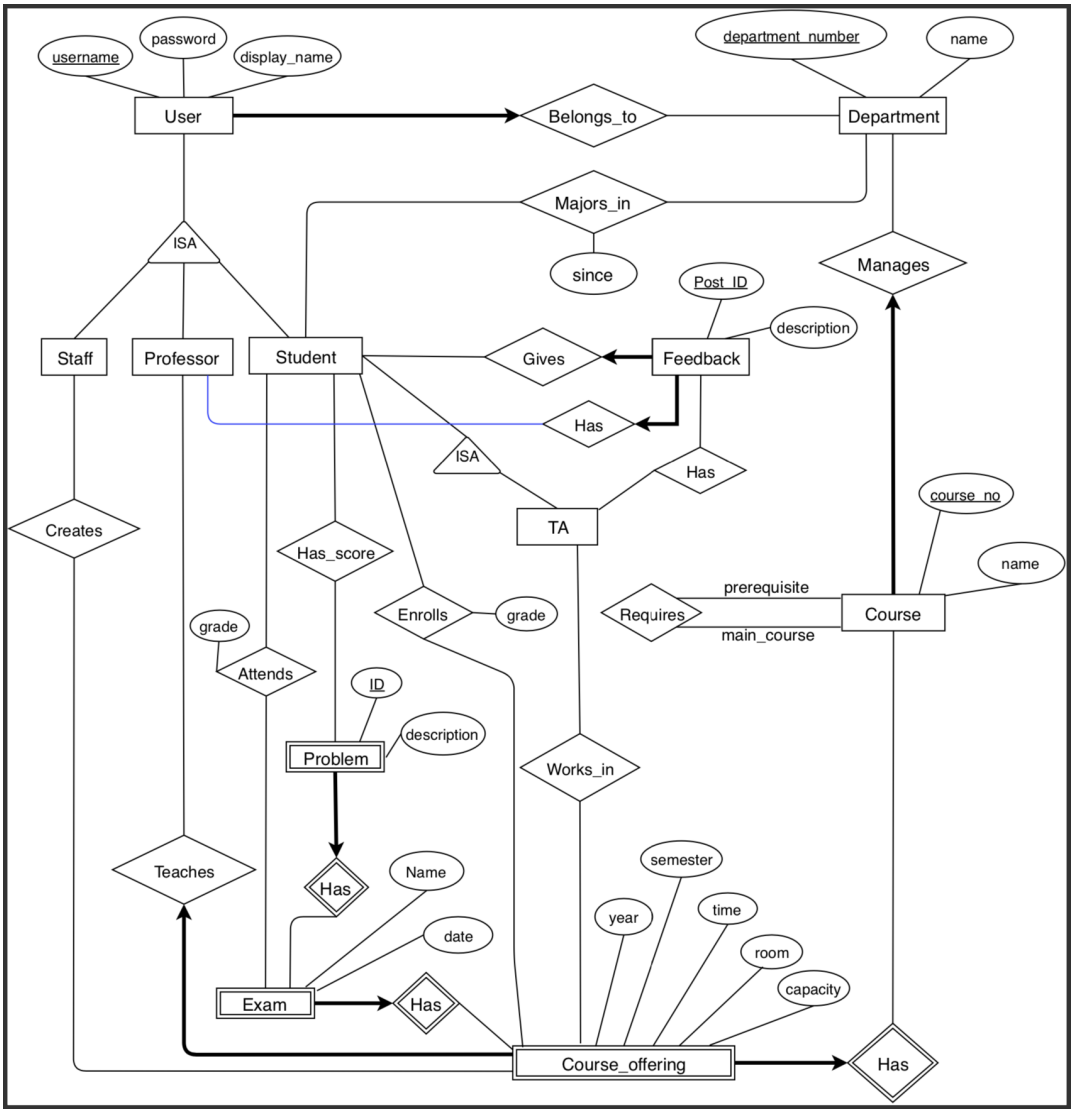


Figure: E/R diagram of **TinyHUB**

In the above ER schema there is an entity set `User` which has three types of users with no overlap. So, there are three more entity sets : `Staff, Professor, Student` which forms a specialization of the `User` enitity. Again the `Student` entity set has some students who can work as Teaching Assistant. The `TA` entity denotes those students who works as teaching-assistant in some courses. `User` and `Department` entity sets are related by the `Belongs_to` relationship. As every user must belongs to a department so there is a participation constraint from 'User to Department' relationship. The other major entity sets are : Course and Feedback. There are some weak entity sets also like `Course_offering`, `Exam` and `Problem`. As some courses have prerequisite courses, so there is a self-relationship `Requires` with two different roles. There are some many-to-many relationships between entity sets. e.g. `Majors_in` relationship between `Student` and `Department` entity sets.

## 2 Relational database schema

The relational database schema of the above E/R schema is given below:

- DEPARTMENT (Department_number, Dname);
- STUDENT (Username, Password, Display_name, Dept_ID);
  foreign key `Dept_ID` references DEPARTMENT(Department_number), Unique Key: Display_name;
- PROFESSOR (Username, Password, Display_name, Designation, Dept_ID);
  foreign key `Dept_ID` references DEPARTMENT(Department_number), Unique Key: Display_name;
- STAFF (Username, Password, Display_name, Designation, Dept_ID);
  foreign key `Dept_ID` references DEPARTMENT(Department_number), Unique Key: Display_name;
- MAJORS_IN (Susername, Dept_ID, Since);
  foreign key `Susername` references STUDENT(Username) & `Dept_ID` references DEPARTMENT(Department_number);
- COURSE (Course_no, Cname, Credit_hours, Dept_ID);
  foreign key `Dept_ID` references DEPARTMENT(Department_number);
- COURSE_PREREQUISITE (Main_course_no, prerequisite_course_no);
  Both foreign key `Main_course_no` and `prerequisite_course_no` references COURSE(Course_no);
- QUARTER (QID, Year, Semester );
  Unique Key: (Year, Semester);
- COURSE_OFFERING (Course_no, QID, Time, Room, Capacity, Instructor, Created_by);
  foreign key `Course_no` references COURSE(Course_no),
  foreign key `QID` references QUARTER(QID),
  foreign key `Instructor` references PROFESSOR(Username) and
  foreign key `Created_by` references STAFF(Username);
- STUDENT_ENROLLEMNT (SID, Course_no, QID, Grade);
  foreign key `SID` references STUDENT(Username) and
  foreign keys: (`Course_no, QID`) references COURSE_OFFERING(Course_no, QID);
- EXAM(ExamID, Course_no, QID, Ename, Description, Date);
  foreign keys: (`Course_no, QID`) references COURSE_OFFERING(Course_no, QID);
  Unique key : (Course_no, QID, Ename)
- EXAM_GRADE(ExamID, Sname, Grade);
  foreign key `ExamID` references EXAM(ExamID) and
  foreign key `Sname` references STUDENT(Username);
- PROBLEM(PID, ExamID, Pname, Description);
  foreign key `ExamID` references EXAM(ExamID);
  Unique key: (ExamID, Pname)

- PROBLEM_SCORE(PID, Sname, Score);
  foreign key `PID` references PROBLEM(PID) and
  foreign key `Sname` references STUDENT(Username);
- TA_ASSIGNMENT(Course_no, QID, Sname, Working_hours);
  foreign keys: (`Course_no, QID`) references COURSE_OFFERING(Course_no, QID) and
  foreign key `Sname` references STUDENT(Username);
- FEEDBACK_POST(Post_ID, Sname, Course_no, QID, Time, Description);
  foreign keys: (`Sname, Course_no, QID`) references STUDENT_ENROLLEMNT (SID,Course_no, QID);
- TA_FEEDBACK(PID, TAname);
  foreign key `PID` references FEEDBACK_POST(Post_ID) and
  foreign key `TAname` references TA_ASSIGNMENT(Sname);
- PROFESSOR_FEEDBACK(PID, Pname);
  foreign key `PID` references FEEDBACK_POST(Post_ID) and
  foreign key `Pname` references Professor(Username);

## 2.1 Mapping from ER schema to Relational Database schema

From E/R schema, Department entity is mapped to DEPARTMENT relation having the department_number as primary key. As the User entity contains three different types of users having no overlap, this entity is not mapped to any relation. Rather the three other entites having 'isa' relationship with the User entity have been mapped to relation of same name in the relational schema. This design choice actually reduces the search time as there is no overlap among the three types of users. The STUDENT relation contains the attributes : Username, Display_Name, Password and Dept_ID. As the email addresses are used as TinyHub usernames of the users, so Username is designated as the primary key. Dept_ID is a foreign key which references the Department_number attribute in the Department relation. Since Display Name is unique, it has been designated as unique key. Same design choices are followed while converting Staff and Professor entities into STAFF and PROFESSOR relations. There is a many-to-many relationship `Majors_in` between the Student and Department entity. This relationship is mapped into a bridge table named MAJORS_IN which contains the primary key of both STUDENT and DEPARTMENT relation and another attribute 'Since'.

The Course entity is represented by COURSE relation. Course_no is the primary key of this table and Dept_ID attribute is used as a foreign key which refers the DEPARTMENT table. This foreign key constraint is used because there is a participation constraint from Course entity to Department entity. Every course must belong to a department according to the problem specification. Here, I assume that every course belongs to exactly one department. The 'Requires' self-relation of the 'Course' entity is mapped to COURSE_PREREQUISITE table. Both Main_course_no and prerequisite_course_no are foreign keys referencing the Course_no attribute of the COURSE table. An important design choice is made to represent the 'Course_offering' weak entity in relational schema. As different professors may teach the same course in different semesters, so I create a relation by using some attributes from the Course_offering entity. This new relation is QUARTER having attributes: QID, Year and Semester. QID is an artificial key which is introduced to identify different semester. For every semester in every year there should be one QID denoting that semester. To ensure this I make the combination of Year and Semester attributes unique. The other attributes of the Course_offering entity is used to map this entity to the COURSE_OFFERING relation where foreign key Course_no references COURSE(Courseno), QID references QUARTER(QID), Instructor references PROFESSOR(Username) and Created_by references STAFF(Username);

A student can enroll into a new course offering if he/she meets the prerequisite of that course which can be easily checked by using the COURSE_PREREQUISITE and the students' previous course records. To store the students' course history the 'Enrolls' relationship is mapped to STUDENT_ENROLLEMNT relation. This is a bridge table which denotes the many-to-many relationship between Student and Course_offering entities. The STUDENT_ENROLLEMNT table contains the primary key of both the corresponding entity , e.g.

SID references STUDENT(Username) and (Courseno, QID) references COURSE_OFFERING(Courseno, QID). The table has another attribute Grade which denotes the students grade in that enrollment record.

The Exam weak entity is represented by EXAM relation having an artificial primary key ExamID. This refers to a specific exam of a specific course offering, e.g. Midterm1 exam for the CSE 560 course offering of FALL 2019. To ensure this the combination of the attributes (Course_no, QID, Ename) is make unique. The 'Attends' relationship and the attribute 'grade' is represented by the relation EXAM_GRADE. The 'Problem' weak entity is mapped to the PROBLEM relation and the 'Has_score' relationship between Student and Problem entity is mapped to the table PROBLEM_SCORE.

'TA' entity is not mapped to any table in the relational schema, rather the relationship 'Works_in' between TA ans Course_offering is mapped to a table TA_ASSIGNMENT. Each record in this table denotes a TA assigned for a particuler course_offering. The table has attributes: Course_no, QID, Sname, Working_hours. The Feedback entity is mapped to FEEDBACK_POST where Post_ID is an artificial key introduced to identify each feedback post uniquely. Here I assume that in a single feedback post a student can give feedback for multiple TAs, so there is a many-to-many relationship between TA and Feedback entites. This is represented by another relation TA_FEEDBACK containg atrributes PID and TAname. PID references FEED_BACKPOST(PostID) and TAname references TA_ASSIGNMENT(Sname). Same choices are followed for PROFESSOR_FEEDBACK table.

## 2.2 Satisfaction of all given requirements

### 2.2.1 User Management

Users can be of three types: Students, Professors, and Staff. In my relational schema, there are three different relation for three different users. It is mentioned that one email address can be used to create a maximum of one user and email addresses are used as the TinyHub usernames of the users. So, using Username as the primary key of the relations STUDENT, STAFF and PROFESSOR enforces this. It is also mentioned that one account, i.e. one username, has only one display name, and one display name corresponds to only one username. Setting Display_Name as a unique key in the STUDENT, STAFF and PROFESSOR tables enforces this requirement. These tables also stores the password for the users, which can be used at the time of login. Students can have multiple majors which is satisfied using the MAJORS_IN tables. For example, if a student has 3 different majors then there should be three corresponding records in the MAJORS_IN tables. Thus all the requirements of user management is satisfied by the given relational schema.

### 2.2.2 Course Management

The department-course management requirements are satisfied through multiple relations. The COURSE table contains all the information related to a course. Course_no attribute denotes the unique number of a course and dept_ID refers the department of that course. It is mentioned that different professors may teach the same course in different semesters. This means that a course has a offering in a particular semester with a specific instructor. To address this situation I have designed two tables: COURSE_OFFERING and QUARTER. For every semester there is a record in the QUARTER table which is identified by artifical priamry key QID. When a course is offered in a semster an entry is made into the COURSE_OFFERING table. In that table, QID refers the semester and Course_no refers the actual course that is being offered. Also there are other attributes: 'Capacity' (a student can enroll into this offering if capacity is not full), 'Instructor' references Username of the PROFESSOR table, 'Created_by' references STAFF(Username). As a course can have zero or more prerequisites so I have a relation COURSE_PREREQUISITE which has two atrributes : one denotes the main course and the other denotes the prerequisite course. So, if a course hase three prerequisite courses, then there there should be three corresponding entry into this table. There are many TAs in a course and all TAs must be student. I have a relation TA_ASSIGNMENT which has attributes: (Courseno, QID) references COURSE_OFFERING(Courseno, QID); this indicates a particular course offering, Sname references STUDENT(Username) which indicates the student who is assigned as a TA in that particular course offering. Thus all the requirements are satisfied.

#### 2.2.3 Student-Course relationship management

A student can enroll in a course only if that student passes all the prerequisite courses, it is being offered by a department they are majoring in,and the capacity of the course is not full. All these are satisfied by the relational schema. From the STUDENT_ENROLLMENT we can easily get the course history of a user and COURSE_PREREQUISITE we can get the prerequisite courses of that specific course. Now we can easily verify if the student has completed all the prerequisite courses or not. From the COURSE table we can know which department is offering that course and from the MAJORS_IN table we can get the moajros of a student. Then we can easily check if the course is being offered by the departments a student is majoring. The information of capacity of that course can be found from COURSE_OFFERING relation. Thus, all the requirements for student enrollment in a course is satisfied. The grade of a student in a particular course offering is entered into the STUDENT_ENROLLEMNT table. Every record in this table has the grade information of a student in a particular course offering. Students can post feedback for the instructor or TAs of the course in which they are enrolled. I design one relation FEEDBACK_POST which contains all the feedbacks. But I also need to ensure that the feedback post from a student is related to the course he/she is enrolled. In the FEEDBACK_POST table there is a foreign key constraint which ensures that every post is valid that means students give feedbacks for their enrolled courses. The foreign keys: (Sname, Courseno, QID) references STUDENT_ENROLLEMNT (SID,Courseno, QID). So whenever a feedback post is posted by a student it is entered into the FEEDBACK_POST table if the foreign key contraints is satisfied. The TA_FEEDBACK table contains all the feedback of TAa and PROFESSOR_FEEDBACK table contains all the feedback of instructors. The EXAM table has all the exam records of a course offering and the EXAM_GRADE relation has the exam grades of every student in an exam of a particular course. Finally, the PROBLEM table has all the problems of an exam and PROBLEM_SCORE stores the score of a student in a particular problem of an exam. Thus, all the requirements are satisfied by the relational schema.

## 3 Further Discussion

The advantage of my design is that there is a clear distinction among the Students, Staffs and Professors, since they are separate entity-types. This design also accommodates the existence of Teaching-Assistants without creating any separate entity. The TA_ASSIGNEMNET table actually stores the Username of the student who are wokring as TA. The design also identifies all the weak entities and their specializations. Nested queries, if required, can be executed very fast as the queries would be on key attributes only. The design also eliminates the data redundancy. Only the primary keys of different relations are stored outside the relations and the data of a relation can be accessed by joining the relations using the reference. Also, by introducing artificial primary key in some relations, I avoided making some entities as a weak entity set. One disadvantage of the design is that there are some composite primary key in some relations which will make the joining slow in some cases.