PROJECT 1

CSE560: Data Models and Query Language

Submitted by:

Team Members :

Rohith Kumar Poshala
UB# 50320370
Mail id :rposhala@buffalo.edu

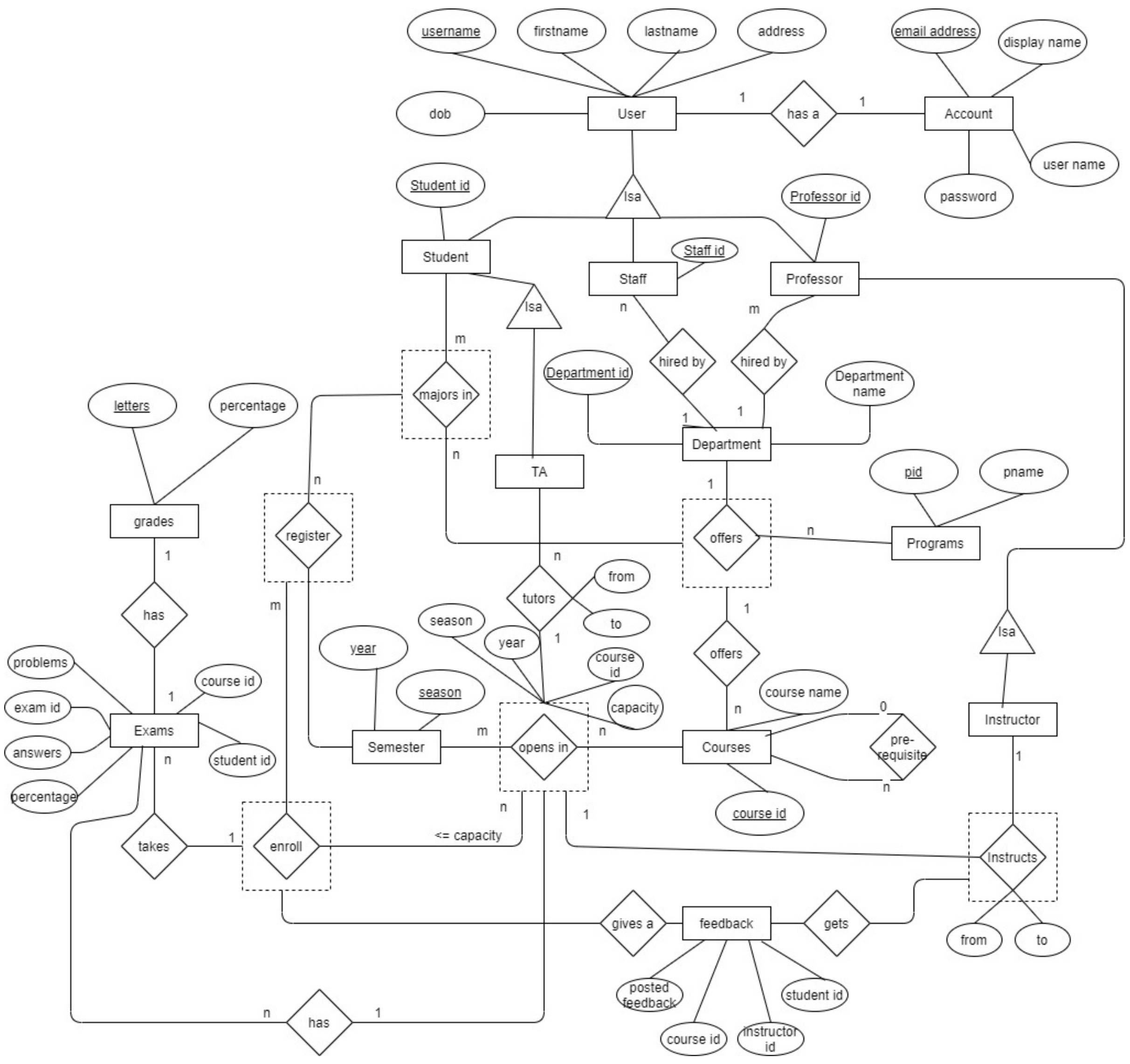
Sreekar Guggilam
UB# 50318495
Mail id :sreekarg@buffalo.edu

We have designed and implemented the database schema for TinyHub, which is a course enrolment website. It provides simple functions, main functions of TinyHub are the following:

➢ User Management : Storing details of different users in an account.
➢ Department management : Professors, Staff and programs managed by Department.
➢ Course management : Managing details of pre-requisite courses, TA's, Instructor and semester in which course is offered and students enrolled.
➢ Student Course management : Recording Student activities such as registering , enrolling , giving exams , getting grades for the course taken.

<u>E/R Schema</u>:

The E/R model for this was plotted as follows:



User Management :

- User entity has attributes username, first name, last name, address, date of birth where username is used as a primary key.
- User entity have **has a** relationship with another entity Account where every user has an account(One to One cardinality).
- Entity Account has attributes email address, password, display name, username where email address is used as a primary key.
- User Entity has **ISA** relationship with Student , Staff, Professor (as child entities).
- Student entity has an attribute Student id.
- Staff entity has an attribute Staff id.
- Professor entity has an attribute professor id.

Department Management :

- Department entity has attributes department id, department name where department id is used a primary key.
- Department has **hired by** relationship with Staff entity with a **One to Many** cardinality and every staff entity should be associated with a department
- Department has a **hired by** relationship with Professor entity with a **One to Many** cardinality and every professor should be associated with a department.
- Department entity has **offers** relationship with programs Entity with a **One to Many** cardinality and every program should be associated with a department.
- Program entity has attributes program id, program name where program id is used a primary key.
- An aggregated entity has been created for relationship **Offers** which has majors in relationship with the Student entity with a **Many to Many** cardinality where a student majors in relation holds with the Offers aggregated entity only when the relation between department and program holds.
- So student can only pursue the programs offered by the department in which he is majoring in.

Course management :

- Entity Courses has attributes Course id , Course name where course id is used as a primary key.
- Courses entity is has **offers** relation with the aggregated entity of Offers relation (between department and program entity) with **Many to One** cardinality which implies that those courses are offers by the program only if the those programs are offered by the department.
- Entity Semester has attributes year and season where both are used as composite key.
- Entity Course has **Opens in** relation with Entity Semester with **Many to Many** relationship.
- Entity TA is created with a **ISA** relationship with Student Entity(Parent Entity).

- An aggregated entity is made of Opens in relation with attributes course id, year, season, capacity where course id, year, season is considered as a composite primary key.
- TA entity has **tutors** relationship with the Aggregated Entity of Opens in relation with **Many to One** cardinality and tutors relation has attributes from and to with it.
- Entity Instructor has been created with **ISA** relationship with Professor entity(parent entity).
- Aggregated Entity of Opens in relation has Instructs relationship with Instructor entity with **One to One** cardinality and Instructs relation has attributes from and to with it.
- And a recursive relation pre-requisites to the entity course was created to get all the pre-requisites of a particular course.
- A aggregated entity has been made of **Majors in** relation and is connected to Semester entity with a **register** relation with **Many to Many** cardinality where a student can only register if he Majors in a department.

Student Course management :

- A aggregated entity of **Register** relation is created and connected to aggregated entity of Opens in relation through a **Enroll** relation with a degree constraint based on capacity where the relation only holds if the entries are less than the value of attribute capacity and only when the relation register exists between majors in and semester. And it has a **Many to Many** cardinality.
- As relation enroll is connected to entity register, implies that relation enroll holds true only when the student majors in department which offers the course he has registered in.
- The recursive relation of pre-requisite to courses connected to entity Opens in enables us to check if the student has passed the required pre-requisite courses of the enrolled course.
- Entity Feedback has attributes posted feedback, student id, instructor id, course id.
- Aggregated entity of relation Enroll has gives relationship with Entity Feedback which in turn has a gets relationship with aggregated entity of Instructs relation.
- Entity Exams has attributes Exam id, problems, answers, percentage, course id, student id where exam id is used as a primary key.
- Entity Grades has attributes letters, percentage where letters is used as a primary key.
- Entity Exams have **has a** relationship with grades entity with **One to One** cardinality.
- Aggregated entity of enroll relation has **takes** relationship with entity Exams with **Many to Many** cardinality.
- Exams entity is connected to Opens in entity with **has a** relationship with **One to Many** cardinality as a course can have more than one exams.

Relational Database Schema:

The schema consists of following tables :

- USER
- ACCOUNT
- PROFESSOR
- STUDENT
- STAFF
- DEPARTMENT
- PROGRAMS
- INSTRUCTOR
- COURSES
- PRE_REQUISITES
- SEMESTER
- TA
- OFFERS
- MAJORSIN
- REGISTER
- ENROLL
- EXAMS
- FEEDBACK
- GRADES
- OPENS_IN
- INSTRUCTS

Schema :

1. USER (username,first_name,last_name,address,dob)
   Primary key (username)
2. ACCOUNT(email_id, display_name, password,username)
   Primary key(email_id)
   FK(username) References USER(username)
3. STUDENT (Student_id, username)
   Primary key(Student_id)
   FK(username) References USER(username)
4. PROFESSOR (Professor_id, username, department_id)
   Primary key(Professor_id)
   FK(username) References USER(username)
   FK(department_id) References DEPARTMENT (department_id)
5. STAFF(staff_id,username,department_id)
   Primary key(staff _id)
   FK(username) References USER(username)
   FK(department_id) References DEPARTMENT (department_id)
6. DEPARTMENT(department_id,department_name)
   Primary key(department_id)
7. PROGRAMS(program_id,program_name,department_id)
   Primary key(program_id)
   FK(department_id) References DEPARTMENT (department_id)
8. INSTRUCTOR(Instructor_id)

Primary key(Instructor_id)
FK(Instructor_id) References DEPARTMENT (Instructor_id)

9. COURSES(course_id,course_name,department_id)
   Primary key(course_id)
   FK(department_id) References OFFERS (department_id)
10. PRE_REQUISITES(course_id,pre_course_id)
    Primary key(course_id,pre_course_id)
    FK(course_id) References COURSES (course_id)
    FK(pre_course_id) References COURSES (course_id)
11. SEMESTER(year,season)
    Primary key(year,season)
12. TA(ta_id ,from,to,course_id,year,season)
    Primary key(ta_id, course_id,year,season)
    FK(ta _id) References STUDENT (student _id)
    FK(course_id,year,season) References OPENS_IN (course_id,year,season)
13. EXAMS(Exam_id,problems,answers,course_id,percentage,student_id)
    Primary key(Exam_id)
    FK(student _id) References ENROLL (student _id)
    FK(course _id) References OPEN_IN (course _id)
14. OFFERS(department_id,program_id)
    Primary key(department_id,program_id)
    FK(department_id) References DEPARTMENT (department_id)
    FK(program_id) References PROGRAMS (program_id)
15. MAJORSIN(Student_id,department_id)
    Primary key(Student_id,department_id)
    FK(Student_id) References STUDENT (Student_id)
    FK(department_id) References OFFERS (department_id)
16. REGISTER(Student_id,year,season)
    Primary key(Student_id,year,season)
    FK(Student_id) References MAJORSIN (Student_id)
    FK(year,season) References SEMESTER (year,season)
17. OPENS_IN(course_id,capacity,year,season,student_id)
    Primary key(course_id ,year,season)
    FK(course_id) References COURSES (course_id)
    FK(year,season) References SEMESTER (year,season)
18. ENROLL(Student_id,course_id,year,season)
    Primary key(Student_id,course_id,year,season)
    FK(Student_id) References REGISTER (Student_id)
    FK(course _id,year,season) References OPENS_IN (course _id,year,season)
19. FEEDBACK(posted_feedback,student_id,course_id,instructor_id,year,season)
    Primary key(student_id,course_id,instructor_id,year,season)
    FK(student_id, course_id,year,season) References ENROLL (student_id, course_id,year,season)
    FK(instructor _id) References INSTRUCTS(instructor _id)
20. INSTRUCTS(Instructor_id,from,to,course_id,year,season)
    Primary key(Instructor_id,course_id,year,season)
    FK(course _id,course,year) References OPENS_IN (course _id,year,season)
    FK(Instructor _id) References INSTRUCTOR (Instructor _id)
21. GRADES(letters,percentage)
    Primary key(letters)

FK(percentage) References EXAMS (percentage)

Schema Justification:

- A ISA relationship has been used for Student, Staff, Professor with User as requirement says User can be of above three types.
- In relational database we declared email id, password, display names as varchar for account entity.
- We declared email id, display name as unique keys, which satisfies the condition of one username has only one display name, one display name corresponds one username and display name can be null ,so not null constraint was not given.
- We made email id as primary key, so the system will allow only one email address to be used for one user.
- We made Hired_by relation between Staff, Professor and Department and one more relation MajorsIn between Student and Department such that we can achieve the requirement of each user belongs to department.
- We made MajorsIn as Many to Many relationship, from which we can satisfy the requirement of student can have multiple majors.
- We created a table Course which will have attributes like name of the course, department id of the course, capacity and course id. We made a relation Offers with aggregated entity of relation Offers (between Department and programs) so that we can get the information about the course to which department it belongs to and if a course exist there should always be a department – program relation. So, student can pursue different programs and only can pursue a program which belongs to the department he majors in.
- We made a relation IS-A between Student and TA so that we can achieve the requirement of a TA must be a student.
- We made a relation tutors between TA and the aggregated entity of opens in relation between course and semester, such that we can achieve the requirement that a course can have many TAs and different TAs in different semesters. As we made Many to One relation for tutors, we can say that many numbers of TAs will be for one course.
- The attributes from and to of TA can provide us information about duration of a particular TA ,so it will satisfy the condition that it will be possible to change TAs during the semester.
- Semester entity has been created with a composite primary key with year and season.
- With a opens in relation courses and semester is connected with intermediate table which provides us the flexibility and reduces the redundancy of course and semester combinations. So a course can be opened in different semesters but not have to be open in every semester.
- We made a relation IS-A between Professor and instructor so that we can achieve the requirement of a Instructor must be a professor
- We made a relation instructs between instructor and the relation between course and semester, such that we can achieve the requirement that different professors may teach the same course in different semesters. As we made One to One relation for instructs, we can say that one instructor teaches the one course.
- The attributes from and to of instructs can provide us information about duration of a particular instructor, so it will satisfy the condition that it will be possible to change instructor during the semester.

- We made a recursive relation Prerequisites which can fetch all the pre-requisite courses of a particular course.
- We assigned one attribute capacity to the aggregated entity of opens in relation table which updates the current status of the course strength.
- For the requirement, it is being offered by a department they are majoring in is achieved by the relation Majors_In between student and department.
- A register relationship is created between majors in and semester and entity enroll entity is created which implies that a student can only register if he majors in and a student can only enroll if he registers.
- For feedback, we made a relation between Enroll and instructs to get all the feedbacks given by each student to each professor. In Enroll we get all the students who enrolled to that particular course in particular semester. At instructs relation we have the professor names who is teaching that particular course in particular semester which student enrolled will be stored. So we can be sure that students are giving feedback to the relevant person.
- Grades entity has letters as primary key and is connected to Exam with an attribute percentage. So student will be given grades based on the overall percentage of the student after a course is done.
- Exams has problems, answers and percentage as attribute which gives us the information of number of problems and scores/percentage of the those problems.
- We have an entity Exam which has an attribute Exam id. This entity connected to the relation between Course and Semester, and Enroll to get a particular exam for a particular course. such that we can get the information about which student attended which exam and what is his grade in that exam.

Further Discussion:

Advantages:
This schema has been designed with very low redundancy.
In this schema we put less burden on programmer as most of the calculations and checking are done in database level itself.

Disadvantages:
We couldn’t implement some functions of the above E/R diagram in relational database. For example, we can’t put some of the constraints which are expected in the model such as we were unable to implement the condition that before enrolling to a course student must pass in all pre-requisites of that course.