PROJECT 1
CSE560: Data Models and Query Languages

Submitted by: Rohit Aggarwal

UB Person Number: 50321371

Email ID: rohitagg@buffalo.edu

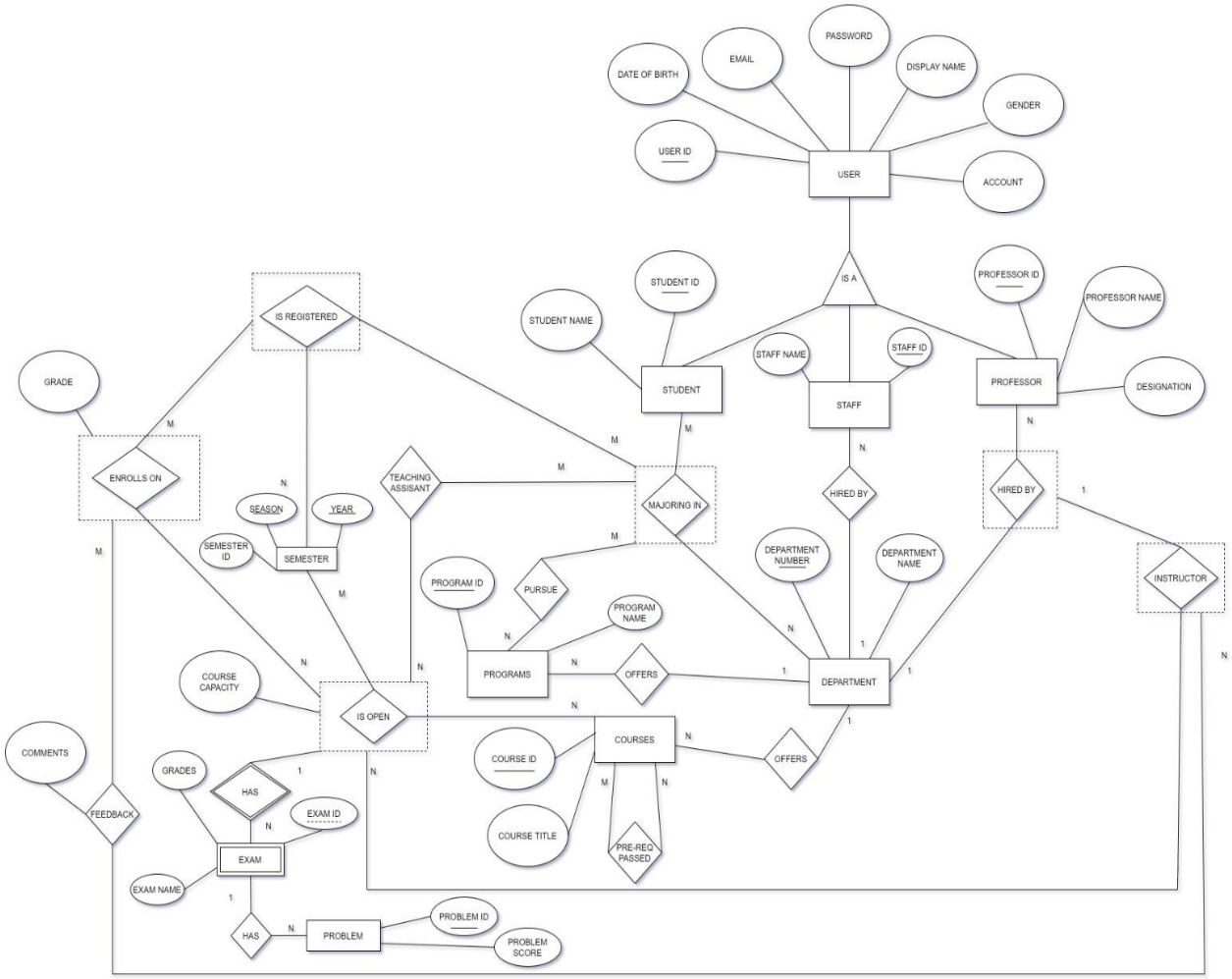
Introduction

The objective is to design and implement a database schema for TinyHub, a course enrolment website. The main functions of TinyHub are listed below:

- ***User Management:*** Users signing up their accounts with their email address and password
- ***Department Management:*** Departments offering different programs and the ability of students to major in and pursue different departments and programs, respectively
- ***Course Management****:* Departments offering different courses, courses being open in different semesters, and the ability of students to register in different semesters
- ***Student-Course Management:*** Ability of students to enrol in different courses provided they've satisfied different requirements, each course consisting of different exams, and students being graded for each course they complete

E/R Schema

The Entity-Relationship (ER) schema that satisfied all the necessary requirements is as shown below:



User Management

- The entity **User** has seven attributes – User_ID (primary key), Email, Password, Display name, Account, Date_of_birth and Gender. User_ID acts as a foreign key for each of the entities, **Student**, **Professor** and **Staff**
- A user signs in their accounts using their email address and by setting up their password
- Each account is registered by only one email address
- A user can be any one of three entities, namely a **Student**, **Professor** or **Staff** and this is represented by an **IS-A** hierarchy
- The entity **Student** has four attributes - Student_ID (primary key), Student_Name, Major_Dept and Semester_ID. Student_ID, major_dept and semester_ID act as foreign keys for the entities **User, Department** and **Semester,** respectively
- The entity **Professor** has four attributes - Professor_ID (primary key), Professor_Name, Designation and Hired_by_dept. Professor_ID and hired_by_dept act as foreign keys for the entities **User** and **Department,** respectively
- The entity **Staff** has four attributes - Staff_ID (primary key), Staff_Name, Designation and Hired_by_dept. Staff_ID and hired_by_dept act as foreign keys for the entities **User** and **Department,** respectively

Department Management

- The entity **Department** has two attributes – Department_number (primary key) and Department_name. Department_number acts as a foreign key for each of the entities **Student, Professor, Staff, Majored_Department** and **Pursued_Program**
- Since each professor and each staff can be **hired** by only one department, and each department can have multiple professors and staffs, the relationship is **One-to-Many**
- The entity **Program** has three attributes - Program ID (primary key), Program Name and Dept_number. Dept_number acts as a foreign key for the entity **Department**
- Since each department can **offer** multiple programs and each program can only belong to one department, the relationship is **One-to-Many**
- Since each student can **major in** multiple departments and each department can be majored by many students, the relation is **Many-to-Many**
- The entity **Majored_Department** has two attributes – Student_ID and Department_Number, both of which act as foreign keys for the entities **Student** and **Department,** respectively
- Students can **pursue** different programs, provided they major in the department offering that programs. Hence, the relationship **Majoring In** is represented as an aggregation
- The entity **Pursued_Program** has three attributes – Student_ID, Department_Number and Program_ID, all of which act as foreign keys for the entities **Student**, **Department** and **Program,** respectively

Course Management

- The entity **Course** has four attributes - Course_ID (primary key), Course title, Department_number and Semester_ID. Department_number and Semester_ID act as foreign keys for the entities **Department** and **Semester,** respectively, while Course_ID acts as a foreign key for each of the entities, **Opened_Course, Pre_Requisite_Course** and **Exam**
- Since each department offers many courses and since each course can be offered by only one department, the relationship is **One-to-Many**
- The entity **Semester** has three attributes – Year and Season, which together act as a composite primary key, and Semester_ID. Semester_ID acts as a foreign key for each of the entities **Student, Course** and **Pre_Requisite_Course**
- Since many courses can be open in one semester, and one course can be opened in multiple semesters, the relationship is **Many-to-Many**
- The entity **Opened_Course** has four attributes – Is_open, Course_ID, Semester_ID and Course_Capacity. Course_ID and Semester_ID act as foreign keys for the entities **Course** and **Semester,** respectively
- Each instructor is a professor and each TA is a student
- Since we assume that each instructor can teach multiple courses opened in one semester and each course opened in different semesters can be taught by only one instructor, the relation is **Many-to-Many**
- Since we assume that each TA can assist multiple courses opened in one semester and each course opened in one semester can be assisted by multiple TAs, the relation is **Many-to-Many**
- Since one course can have multiple pre-requisite courses and each pre-requisite course can be pre-requisite for multiple courses, the relation is **Many-to-Many.** Here, entities within the entity set **Course** have a relation amongst themselves
- The entity **Pre_Requisite_Course** has two attributes – Course_ID and Pre_requisite_course_ID. Course_ID acts as a foreign key for the entity **Course**
- Since each student can register in different semesters and each semester can have multiple students registered, the relation is **Many-to-Many**
- The entity **Registered_Semester** has three attributes – Registration_ID (primary key), Student_ID and Semester_ID. Student_ID and Semester_ID act as foreign keys for the entities **Student** and **Semester,** respectively

Student-Course Management

- Certain conditions must be satisfied before a student can **enroll** in a course:
    - Firstly, a student should pass all pre-requisite courses, which is achieved by the **Pre-Req Passed** relationship that exists between entities within the **Course** entity set
    - Secondly, a student must enrol in a course which is open in a semester, provided that it has been registered by the student. This is achieved by the

Majoring In, Is Registered and **Is Open** relationships that are represented as aggregations
    - Thirdly, the course to be enrolled must be offered by the department the student is majoring in. This is achieved by the **Majoring In** aggregation and the **Offers** relationship
    - Finally, the capacity of the course shouldn't have reached its maximum. This can be verified by the Course Capacity attribute of the **Is Open** aggregation
- The entity **Enrolled_Course** has four attributes – Enrolled_Course_ID (Primary key), Course_grade, Student_ID and Course_ID. Student_ID and Course_ID act as foreign keys for the entities **Student** and **Course,** respectively
- Since multiple students of one course can provide **feedback** for one instructor, and multiple instructors get feedbacks from each student of different courses, the relationship is **Many-to-Many**
- The entity **Exam** has four attributes – Exam_ID, Exam_type, Course_ID and Grade. Exam_ID is a unique key and Course_ID acts as a foreign key for the entity **Course.** Since Exam depends on many other entities, it's treated as a weak entity
- Since one course can have multiple exams and each exam is part of one course, the relationship is **One-to-Many**
- The entity **Problem** has three attributes – Problem_ID (Primary Key), Problem_Score and Exam_ID. Exam_ID acts as a foreign key for the entity **Exam**
- Since one exam can have multiple problems and each problem is part of one exam, the relationship is **One-to-Many**
- The entity **Feedback** has three attributes – Comments, Instructor_ID and Enrolled_Course_ID. Instructor_ID and Enrolled_Course_ID act as foreign keys for the entities **Instructor** and **Enrolled_Course,** respectively
- The entity **Instructor** has three attributes – Instructor_ID (Primary Key), Course_ID and Professor_ID. Course_ID and Professor_ID act as foreign keys for the entities **Course** and **Professor,** respectively
- The entity **Teaching_Assistant** has three attributes – TA_ID (Primary Key), Student_ID and Course_ID. Student_ID and Course_ID act as foreign keys for the entities **Student** and **Course,** respectively
- The entity **Finished_Course** has four attributes – Finished_Course_ID (Primary key), Course_grade, Student_ID and Course_ID. Student_ID and Course_ID act as foreign keys for the entities **Student** and **Course,** respectively

Relational Database Schema

The relational database schema consists of the following tables:

- USER
- STUDENT
- PROFESSOR
- STAFF
- DEPARTMENT

- MAJORED_DEPARTMENT
- PROGRAM
- PURSUED_PROGRAM
- COURSE
- SEMESTER
- OPENED_COURSE
- PRE_REQUISITE_COURSE
- REGISTERED_SEMESTER
- ENROLLED_COURSE
- EXAM
- PROBLEM
- COURSE_FEEDBACK
- INSTRUCTOR
- TEACHING_ASSISTANT
- FINISHED_COURSE
- SCORE

Schema:

- USER (user_id, email, password, account, display_name, gender, date_of_birth)
  Here, user_id is the primary key and both email_id and display_name are unique keys.

- STUDENT (student_id, student_name, user_id, major_dept)
  Here, student_id is the primary key and user_id is a foreign key for the USER table.
  FK (user_id) references USER (user_id)

- PROFESSOR (professor_id, professor_name, user_id, designation, hired_by_dept)
  Here, professor_id is the primary key, and user_id and hired_by_dept are foreign keys for the USER and DEPARTMENT tables, respectively.
  FK (user_id) references USER (user_id)
  FK (hired_by_dept) references DEPARTMENT(dept_no)

- STAFF (staff_id, staff_name, user_id, designation, hired_by_dept)
  Here, staff_id is the primary key, and user_id and hired_by_dept are foreign keys for the USER and DEPARTMENT tables, respectively.
  FK (user_id) references USER (user_id)
  FK (hired_by_dept) references DEPARTMENT(dept_no)

- DEPARTMENT (dept_no, dept_name)
  Here, dept_no is the primary key.

- MAJORED_DEPARTMENT (student_id, dept_no)
  Here, student_id and dept_no are foreign keys for the STUDENT and DEPARTMENT tables, respectively.

FK (student_id) references STUDENT(student_id),
FK (dept_no) references DEPARTMENT(dept_no)

- PROGRAM (program_id, program_name, dept_no)
  Here, program_id is the primary key and dept_no is the foreign key for the DEPARTMENT table.
  FK (dept_no) references DEPARTMENT(dept_no)

- PURSUED_PROGRAM (student_id, program_id, dept_no)
  Here, student_id, program_id and dept_no are foreign keys for the STUDENT, PROGRAM and DEPARTMENT tables.
  FK (student_id) references STUDENT(student_id)
  FK (program_id) references PROGRAM(program_id)
  FK (dept_no) references DEPARTMENT(dept_no)

- COURSE (course_id, course_title, dept_no, semester_id)
  Here, course_id is the primary key, and dept_no and semester_id are foreign keys for the DEPARTMENT and SEMESTER tables.
  FK (dept_no) references DEPARTMENT(dept_no)
  FK (semester_id) references SEMESTER (semester_id)

- SEMESTER (semester_id, year, season)
  Here, year and season are the composite primary key.

- OPENED_COURSE (is_opened, course_id, semester_id, course_capacity)
  Here, course_id and semester_id are the foreign keys for the COURSE and SEMESTER tables.
  FK (course_id) references COURSE(course_id)
  FK (semester_id) references SEMESTER (semester_id)

- PRE_REQUISITE_COURSE (course_id, pre_requisite_course_id)
  Here, both course_id and pre_requisite_course_id are foreign keys for the COURSE table.
  FK (course_id) references COURSE(course_id)
  FK (pre_requisite_course_id) references COURSE(course_id)

- REGISTERED_SEMESTER (registration_id, student_id, semester_id)
  Here, registration_id is the primary key, and student_id and semester_id are the foreign keys for the STUDENT and SEMESTER tables, respectively.
  FK (student_id) references STUDENT(student_id)
  FK (semester_id) references SEMESTER (semester_id)

- ENROLLED_COURSE (enrolled_course_id, student_id, course_id)
  Here, enrolled_course_id is the primary key, and student_id and course_id are the foreign keys for the STUDENT and COURSE tables, respectively.
  FK (student_id) references STUDENT(student_id)
  FK (course_id) references COURSE (course_id)

- EXAM (exam_id, exam_name, course_id, grade)
  Here, exam_id is a unique key and course_id is the foreign key for the COURSE table.
  FK (course_id) references COURSE (course_id)

- PROBLEM (problem_id, problem_score, exam_id)
  Here, problem_id is the primary key and exam_id is the foreign key for the EXAM table.
  FK (exam_id) references EXAM (exam_id)

- COURSE_FEEDBACK (comments, instructor_id, enrolled_course_id)
  Here, instructor_id and enrolled_course_id are the foreign keys for the INSTRUCTOR and ENROLLED_COURSE tables.
  FK (instructor_id) references INSTRUCTOR (instructor_id)
  FK (enrolled_course_id) references ENROLLED_COURSE (enrolled_course_id)

- INSTRUCTOR (instructor_id, professor_id, course_id)
  Here, instructor_id is the primary key, and professor_id and course_id are the foreign keys for the PROFESSOR and COURSE tables, respectively.
  FK (professor_id) references PROFESSOR (professor_id)
  FK (course_id) references COURSE (course_id)

- TEACHING_ASSISTANT (ta_id, student_id, course_id)
  Here, ta_id is the primary key, and student_id and course_id are the foreign keys for the STUDENT and COURSE tables, respectively.
  FK (student_id) references STUDENT (student_id)
  FK (course_id) references COURSE (course_id)

- FINISHED_COURSE (finished_course_id, course_grade, student_id, course_id)
  Here, finished_course_id is the primary key, and student_id and course_id are the foreign keys for the STUDENT and COURSE tables, respectively.
  FK (student_id) references STUDENT (student_id)
  FK (course_id) references COURSE (course_id)

- SCORE (score_id, problem_id, enrolled_course_id)
  Here, score is the primary key, and problem_id and enrolled_course_id are the foreign keys for the PROBLEM and ENROLLED_COURSE tables, respectively.
  FK (problem_id) references PROBLEM (problem_id)
  FK (enrolled_course_id) references ENROLLED_COURSE (enrolled_course_id)

Schema Justification

1. Since a user can be any of a Student, Professor or Staff, an IS-A hierarchy exists between Student, Professor and Staff with User
2. Account, password and display names are declared as varchar. Additionally, a check constraint is there to ensure only one of 'Student', 'Professor' or 'Staff' can be given as account, else a new record can't be inserted
3. Display name and email address are unique keys, where display name can be NULL, which satisfy the conditions of an account having an optional unique display name and only one email address being used for one account
4. Department_no is the primary key of department that uniquely identifies a department
5. One-to-many relationship existing between department and each of professor and staff to say each professor and staff can only be hired by one department, and one department can hire multiple professors and staffs
6. One-to-many relationship existing between department and program to say each program is offered by one department, and one department can offer multiple programs
7. Many-to-many relationship existing between department and student to say each student can enrol in multiple departments, and one department can be enrolled by multiple students
8. The relationship Majoring In acting as an aggregation that allows multiple students to pursue different programs, provided they major in the department offering that program. This is represented by a many-to-many relationship
9. One-to-many relationship existing between department and course to say each course is offered by one department, and one department can offer multiple courses
10. Many-to-many relationship existing between course and semester to say each course can be open in multiple semesters, and one semester can be open to multiple courses
11. A distinct combination of season and year acting as the composite primary key of Semester
12. Semester_ID isn't an attribute that can be NOT NULL in the Course entity suggesting that each course needn't be open in each semester
13. The one-to-many and many-to-many relationships existing between course and instructor, and course and TA suggesting that each course has one instructor and multiple TAs, respectively
14. The Instructs and TA relationships between course and professor, and course and student suggesting that each instructor is a professor and each TA is a student, respectively
15. A relationship exists between entities in the entity set Course, which suggests that a course may/may not have multiple pre-requisite courses
16. Many-to-many relationship existing between student and semester to say each student can register in multiple semesters, and one semester can be registered by multiple students

17. The Is Registered and Is Open aggregations suggesting that a student can enrol in a course only if it's open in that semester and that the student has registered in that semester
18. The Pre_requisite passed relationship between entities in the Course entity set saying that a student can enrol a course only if they've passed all pre-requisite courses
19. The Majoring In aggregation between Student and Department saying that a student can enrol a course only if it's offered by the department they're majoring in
20. The Is Open aggregation has an attribute Course Capacity, which checks to see if the capacity of the course the student wishes to enrol in has been met or not
21. Check constraint to verify if the grade awarded to the student after completing the course is one of 'A', 'B', 'C', 'D' or 'F'
22. The feedback relationship between the Enrolls on and Instructs aggregation that says enrolled students can provide feedback to the course's instructor
23. One-to-many relationship existing between course and exam to say each course has multiple exams, and each exam belongs to one course. Also, the Exam entity has an attribute Grade that represents the grade of the exam
24. One-to-many relationship existing between exam and problem to say each exam has multiple problems, and each problem belongs to one exam. The problem entity also has an attribute grade

Further Discussion

Advantages

Some of the requirements/functions we were able to implement were:

- perform certain check constraints on some columns to ensure a new record having incorrect value for that column, can't be inserted
- keeping most of the processing and validation checks at the database level to ensure the programmer isn't burdened much at his/her end
- Enforcement of referential integrity by defining primary/foreign key relationships which will enable us to extract information from multiple tables by joining them
- Extracting the following:
    - information about the department offering a particular course
    - list of courses offered by the department the student is majoring in
    - list of all pre-requisite courses for a particular course
    - list of staffs and professors hired by a particular department
    - list of students majoring in a department and list of departments that a student is majoring in
    - list of programs that a student is pursuing
    - list of courses that a student has enrolled in a particular exam given that the course capacity wasn't full
    - list of students who gave feedback for an instructor of a course
    - list of TAs who assist in a particular course

- list of problem scores for a particular exam of a particular course for each student
- list of semesters that a student has registered in

Disadvantages

Some of the requirements/functions we were unable to implement through the ER diagram were:

- keep a track of the number of students enrolled for each course
- extract the score that a student got for a particular problem in a particular exam
- extract the list of pre-requisite courses that a student has passed for a particular course