CSE 560 Project1

"TINYHUB: A Course Enrollment Website"

By

Samratsinh Dhumal	**50321053**
Pranjal Jain	**50322132**
Tanya Jain	**50321876**
Kavya Anantha Rao	**50320700**
Anchal Sojatiya	**50321238**

University at Buffalo
The State University of New York

Spring 2020

Introduction:

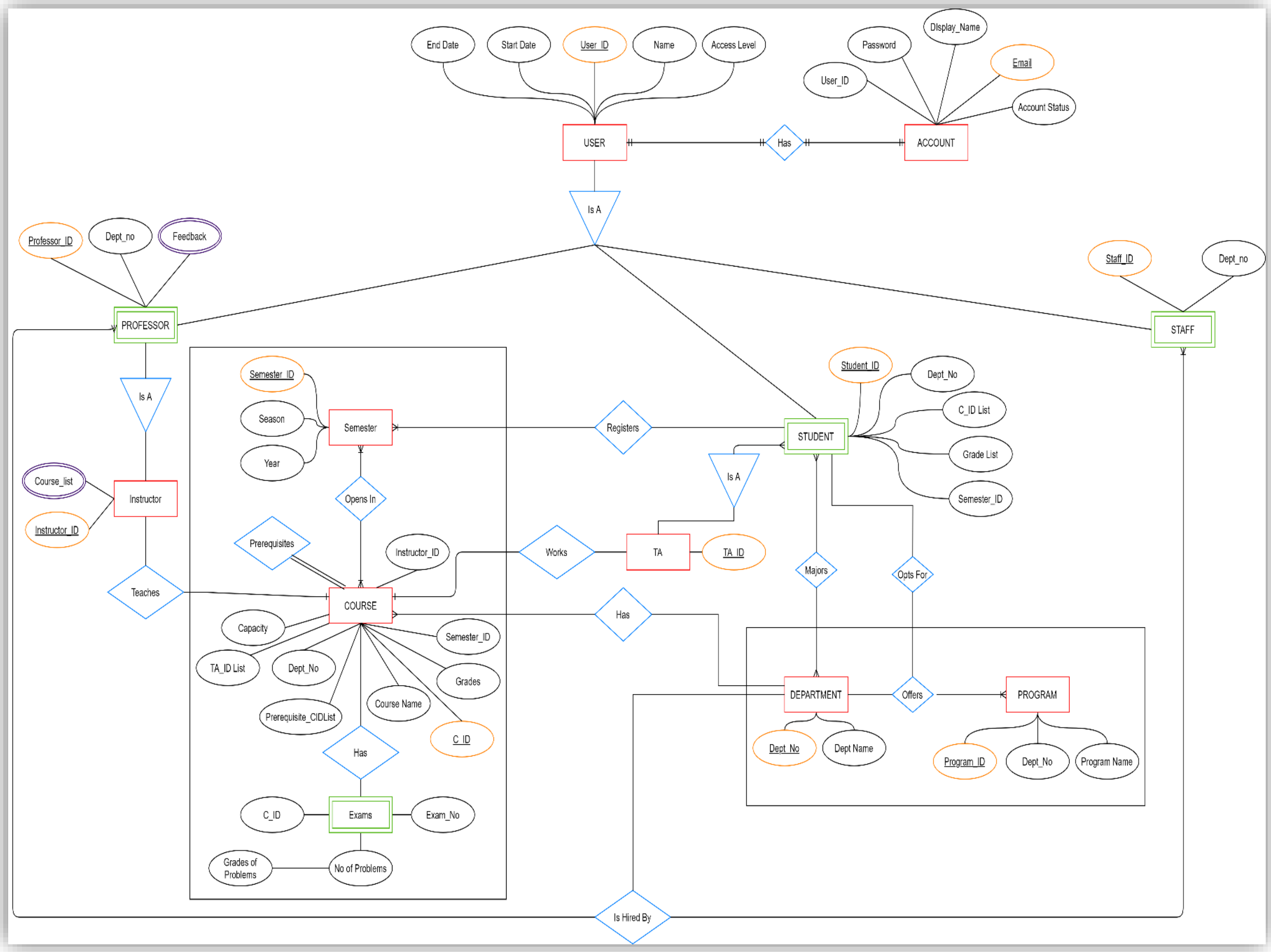
TinyHub is a database system created to maintain the details of the students, professors, departments, programs, courses offered and grades.

Main functions of TinyHub are:

- User management: User sign up, user login/logout.
- Department management: Department functionality.
- Course management: Course and the pre-requisites functionality.
- Student-Course management: Register/ enroll for a course, provide feedback

E/R Schema:

The Entity Relationship diagram for the 'TinyHub' database provides the visualization the data flow across various entities.is shown below:



Tables:

The tables created to accomplish the requirements are according to order of execution:

- User
- Account
- Student
- Professor
- Instructor
- Staff
- Department
- CourseList
- Feedback
- Program
- TeachingAssistant
- Semester
- Course
- Exams
- Prerequisite

Functions:

The system is executed in four stages:

1. **User Management:**
   - The ‘USER’ entity set refers to all the members enrolled in the university. The Users can be broadly classified as entities – STUDENT, PROFESSOR and STAFF (described with a ISA relationship) with each of them requiring signing up to their accounts using their respective email address which is unique to each user.
   - The USER entity has attributes `User_ID`, `User_Name`, `Start_Date`, `End_Date`, `Access_Level`. The User ID is primary key.
   - The Access_Level helps to identify what type of access the user has. We have have 3 access levels as per our type of users.
   - The USER is connected with an ACCOUNT entity with a ‘Has A’ relationship.
   - The ACCOUNT entity has attributes `User_ID`, `Display_Name`, `Email_ID`, `Password`, `Account_Status`. As per the Instructions the Email ID is a Unique and Primary key, The User_ID, display name are also unique.

2. **Department management:**
   - The department management includes the various departments, the professors/staff enrolled in a department, programs offered by each department.
   - The STUDENT entity has attributes `Student_ID`, `Department_No`, `Grade_List`, `Course_ID_List`, `Semester_ID`. The Student_ID is the primary key derived from User_Id. The Dept_No, Semester_ID are Foreign Keys. The C_ID list stores the list of courses the student has taken.

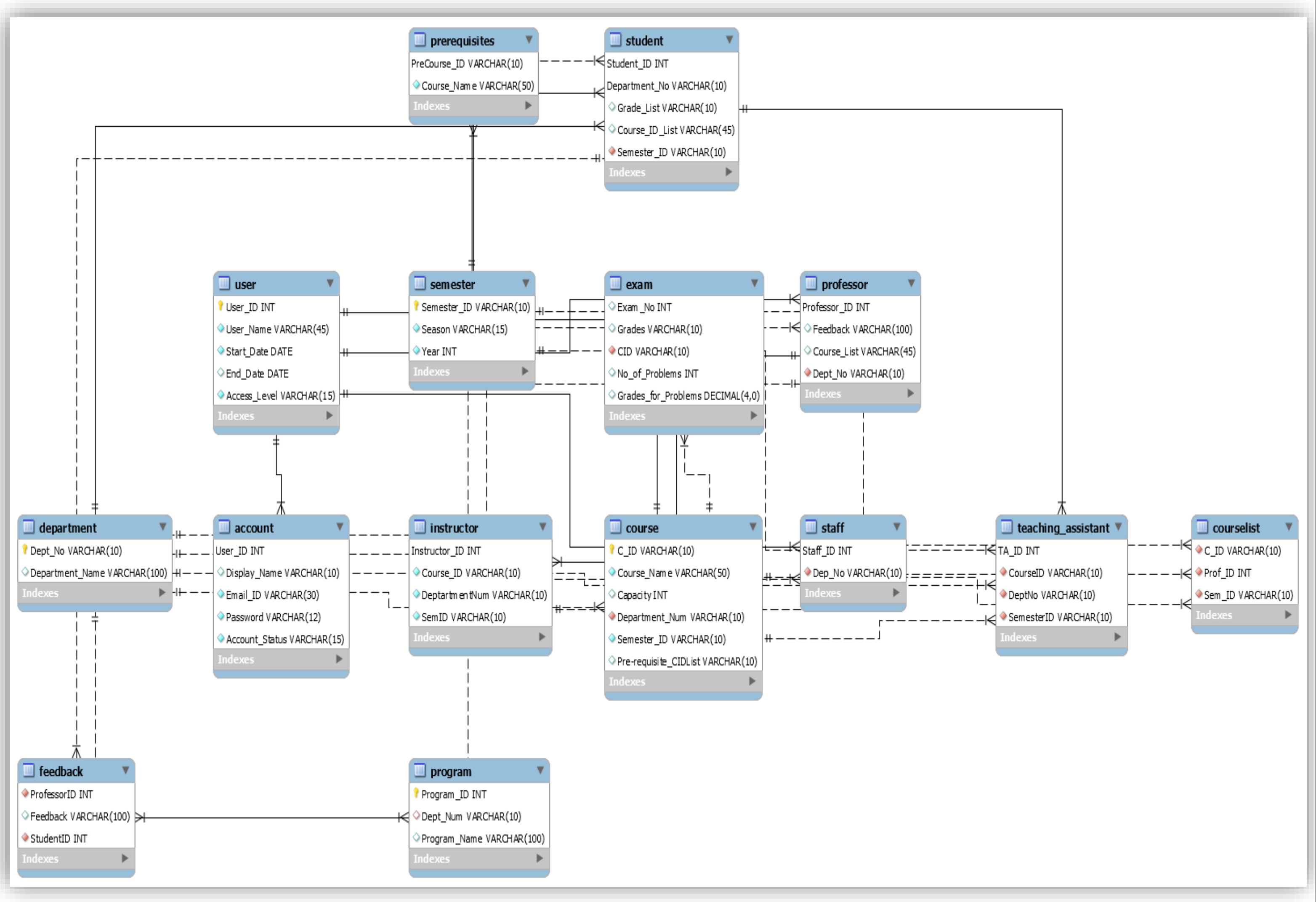
- In this section the PROFFESSOR and STAFF are connected to DEPARTMENT using ‘Hired by’ relationship and one to many cardinality implying that both are hired by one department.
- The PROFESSOR table has attributes `Student_ID`, `Department_No`, `Grade_List`, `Semester_ID`. These attributes help them connect with department and semester tables.
- The STAFF table is also derivation of the USER table which has attributes as `Staff_ID`, `Dep_No`.
- The DEPARTMENT table has attributes `Dept_No` and `Department_Name’.
- STUDENT is joined to DEPARTMENT by many-to-many ‘Majors In’ relationship, showing that the student can major in different departments.
- DEPARTMENT and PROGRAM are considered as one entity by putting an aggregation box. The aggregation here shows that the STUDENT can Opt for multiple programs but he must major in the DEPARTMENT which offers that program.

3. **Course management:**
   - This section focuses on the courses offered by the department, pre-requisites if any, when the courses may be opted for (which semester), the teaching assistants for each course.
   - The COURSE entity has multiple attributes as mentioned in the ER.
   - Here STUDENT ‘Registers’ in SEMESTER with one to many relation showing that that one student can register in multiple semesters. Each DEPARTMENT ‘Has’ multiple COURSES is shown in one to many relation.
   - The TEACHING ASSISTANT entity is derived from the STUDENT and the INSTRUCTOR entity is derived from the PROFESSOR.COURSE entity is connected to itself with many to many cardinality using PREREQUISITE relationship.
   - The INSTRUCTOR has COURSE_LIST multivariate attribute which is treated as new table and the details of the courses the instructor teaches are stored here.
   - Also the prerequisite courses are shown by self-joining the course by ‘Prerequisite’ relationship.

4. **Student-Course management:**
   - Here the details pertaining to each course, the grades obtained on the exam, feedback provided to the professors.
   - The aggregation is used for SEMESTER and COURSE showing that anyone must register in semester first before enrolling in any course.
   - The capacity can be checked using the ‘Capacity’ attribute in the COURSE entity. Grades are stored in the ‘Grades’ attribute.
   - The Self relation of COURSE using PREREQUISTE shows that the student should pass all the prerequisite courses.
   - Also the PROFESSOR has the FEEDBACK multivariate attribute for which we have created a new table wherein the feedbacks given to the professor are stored.

- EXAMS is entity where the number of exams, number of problems per exam and grades of each problem are stored. Using all the grades of problem and number of exams total grade of the course is stored in the 'Grades' attribute in the COURSE entity. Exam has composite attributes for no of problems and grades.
- The COURSE is connected to DEPARTMENT using 'Has A' relationship showing that the department must have that course.

The SQL EER diagram is as below:



Scripts:

The Scripts include the schematic representation of the ER in form of queries.
The Global constraints of each table including the Primary and Foreign Keys are mentioned in the scripts.

```
-- Schema Tinyhub
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `Tinyhub` ;
USE `Tinyhub` ;

-- -----------------------------------------------------
-- Table `Tinyhub`.`User`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Tinyhub`.`User` (
  `User_ID` INT NOT NULL,
  `User_Name` VARCHAR(45) NOT NULL,
  `Start_Date` DATE NOT NULL,
  `End_Date` DATE NULL,
  `Access_Level` VARCHAR(15) NOT NULL,
  PRIMARY KEY (`User_ID`))

-- -----------------------------------------------------
-- Table `Tinyhub`.`Account`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Tinyhub`.`Account` (
  `User_ID` INT NOT NULL,
  `Display_Name` VARCHAR(10) NULL,
  `Email_ID` VARCHAR(30) NOT NULL,
  `Password` VARCHAR(12) NOT NULL,
  `Account_Status` VARCHAR(15) NOT NULL,
  UNIQUE INDEX `Display_Name_UNIQUE` (`Display_Name` ASC) VISIBLE,
  INDEX `User_ID_idx` (`User_ID` ASC) VISIBLE,
  UNIQUE INDEX `User_ID_UNIQUE` (`User_ID` ASC) VISIBLE,
  UNIQUE INDEX `Emial_ID_UNIQUE` (`Email_ID` ASC) VISIBLE,
  PRIMARY KEY (`User_ID`),
  CONSTRAINT `User_ID`
    FOREIGN KEY (`User_ID`)
    REFERENCES `Tinyhub`.`User` (`User_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

```
-- -----------------------------------------------------
-- Table `Tinyhub`.`Department`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Tinyhub`.`Department` (
  `Dept_No` VARCHAR(10) NOT NULL,
  `Department_Name` VARCHAR(100) NULL,
  PRIMARY KEY (`Dept_No`))

-- -----------------------------------------------------
-- Table `Tinyhub`.`Professor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Tinyhub`.`Professor` (
  `Professor_ID` INT NOT NULL,
  `Feedback` VARCHAR(100) NULL,
  `Course_List` VARCHAR(45) NULL,
  `Dept_No` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`Professor_ID`),
  INDEX `Dept_No_idx` (`Dept_No` ASC) VISIBLE,
  UNIQUE INDEX `Professor_ID_UNIQUE` (`Professor_ID` ASC) VISIBLE,
  CONSTRAINT `Professor_ID`
    FOREIGN KEY (`Professor_ID`)
    REFERENCES `Tinyhub`.`User` (`User_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `Dept_No`
    FOREIGN KEY (`Dept_No`)
    REFERENCES `Tinyhub`.`Department` (`Dept_No`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)

-- -----------------------------------------------------
-- Table `Tinyhub`.`Staff`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Tinyhub`.`Staff` (
  `Staff_ID` INT NOT NULL,
  `Dep_No` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`Staff_ID`),
  INDEX `Dept_No_idx` (`Dep_No` ASC) VISIBLE,
  CONSTRAINT `Staff_ID`
    FOREIGN KEY (`Staff_ID`)
    REFERENCES `Tinyhub`.`User` (`User_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `Dep_No`
    FOREIGN KEY (`Dep_No`)
    REFERENCES `Tinyhub`.`Department` (`Dept_No`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)

-- -----------------------------------------------------
-- Table `Tinyhub`.`Semester`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Tinyhub`.`Semester` (
  `Semester_ID` VARCHAR(10) NOT NULL,
  `Season` VARCHAR(15) NOT NULL,
  `Year` INT NOT NULL,
  PRIMARY KEY (`Semester_ID`))

-- -----------------------------------------------------
-- Table `Tinyhub`.`Student`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Tinyhub`.`Student` (
  `Student_ID` INT NOT NULL,
  `Department_No` VARCHAR(10) NOT NULL,
  `Grade_List` VARCHAR(10) NULL,
  `Course_ID_List` VARCHAR(45) NULL,
  `Semester_ID` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`Student_ID`, `Department_No`),
  INDEX `Dept_No_idx` (`Department_No` ASC) VISIBLE,
  INDEX `Semester_ID_idx` (`Semester_ID` ASC) VISIBLE,
  CONSTRAINT `Department_No`
    FOREIGN KEY (`Department_No`)
    REFERENCES `Tinyhub`.`Department` (`Dept_No`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `Student_ID`
    FOREIGN KEY (`Student_ID`)
    REFERENCES `Tinyhub`.`User` (`User_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `Semester_ID`
    FOREIGN KEY (`Semester_ID`)
    REFERENCES `Tinyhub`.`Semester` (`Semester_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

```
-- -----------------------------------------------------
-- Table `Tinyhub`.`Program`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Tinyhub`.`Program` (
  `Program_ID` INT NOT NULL,
  `Dept_Num` VARCHAR(10) NULL,
  `Program_Name` VARCHAR(100) NULL,
  PRIMARY KEY (`Program_ID`),
  INDEX `Dept_No_idx` (`Dept_Num` ASC) VISIBLE,
  CONSTRAINT `Dept_Num`
    FOREIGN KEY (`Dept_Num`)
    REFERENCES `Tinyhub`.`Department` (`Dept_No`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)

-- -----------------------------------------------------
-- Table `Tinyhub`.`Course`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Tinyhub`.`Course` (
  `C_ID` VARCHAR(10) NOT NULL,
  `Course_Name` VARCHAR(50) NOT NULL,
  `Capacity` INT NULL,
  `Department_Num` VARCHAR(10) NOT NULL,
  `Semester_ID` VARCHAR(10) NOT NULL,
  `Pre-requisite_CIDList` VARCHAR(10) NULL,
  PRIMARY KEY (`C_ID`),
  INDEX `Dept_No_idx` (`Department_Num` ASC) VISIBLE,
  CONSTRAINT `Department_Num`
    FOREIGN KEY (`Department_Num`)
    REFERENCES `Tinyhub`.`Department` (`Dept_No`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)

-- -----------------------------------------------------
-- Table `Tinyhub`.`Feedback`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Tinyhub`.`Feedback` (
  `ProfessorID` INT NOT NULL,
  `Feedback` VARCHAR(100) NULL,
  `StudentID` INT NOT NULL,
  INDEX `Student_ID_idx` (`StudentID` ASC) VISIBLE,
  INDEX `Professor_ID_idx` (`ProfessorID` ASC) VISIBLE,
  CONSTRAINT `StudentID`
    FOREIGN KEY (`StudentID`)
    REFERENCES `Tinyhub`.`Student` (`Student_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `ProfessorID`
    FOREIGN KEY (`ProfessorID`)
    REFERENCES `Tinyhub`.`Professor` (`Professor_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)

-- -----------------------------------------------------
-- Table `Tinyhub`.`Instructor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Tinyhub`.`Instructor` (
  `Instructor_ID` INT NOT NULL,
  `Course_ID` VARCHAR(10) NOT NULL,
  `DeptartmentNum` VARCHAR(10) NOT NULL,
  `SemID` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`Instructor_ID`),
  CONSTRAINT `Instructor_ID`
    FOREIGN KEY (`Instructor_ID`)
    REFERENCES `Tinyhub`.`Professor` (`Professor_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)

-- -----------------------------------------------------
-- Table `Tinyhub`.`CourseList`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Tinyhub`.`CourseList` (
  `C_ID` VARCHAR(10) NOT NULL,
  `Prof_ID` INT NOT NULL,
  `Sem_ID` VARCHAR(10) NOT NULL,
  INDEX `Sem_ID_idx` (`Sem_ID` ASC) VISIBLE,
  INDEX `Prof_ID_idx` (`Prof_ID` ASC) VISIBLE,
  CONSTRAINT `Prof_ID`
    FOREIGN KEY (`Prof_ID`)
    REFERENCES `Tinyhub`.`Instructor` (`Instructor_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `C_ID`
    FOREIGN KEY (`C_ID`)
    REFERENCES `Tinyhub`.`Course` (`C_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `Sem_ID`
    FOREIGN KEY (`Sem_ID`)
    REFERENCES `Tinyhub`.`Semester` (`Semester_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)

-- -----------------------------------------------------
-- Table `Tinyhub`.`Teaching_Assistant`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Tinyhub`.`Teaching_Assistant` (
  `TA_ID` INT NOT NULL,
  `CourseID` VARCHAR(10) NOT NULL,
  `DeptNo` VARCHAR(10) NOT NULL,
  `SemesterID` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`TA_ID`),
  INDEX `C_ID_idx` (`CourseID` ASC) VISIBLE,
  INDEX `Sem_ID_idx` (`SemesterID` ASC) VISIBLE,
  INDEX `Dept_No_idx` (`DeptNo` ASC) VISIBLE,
  CONSTRAINT `TA_ID`
    FOREIGN KEY (`TA_ID`)
    REFERENCES `Tinyhub`.`Student` (`Student_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `CourseID`
    FOREIGN KEY (`CourseID`)
    REFERENCES `Tinyhub`.`Course` (`C_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `SemesterID`
    FOREIGN KEY (`SemesterID`)
    REFERENCES `Tinyhub`.`Semester` (`Semester_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `DeptNo`
    FOREIGN KEY (`DeptNo`)
    REFERENCES `Tinyhub`.`Department` (`Dept_No`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)


-- -----------------------------------------------------
-- Table `Tinyhub`.`Prerequisites`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Tinyhub`.`Prerequisites` (
  `PreCourse_ID` VARCHAR(10) NOT NULL,
  `Course_Name` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`PreCourse_ID`),
  CONSTRAINT `PreCourse_ID`
```

```
    FOREIGN KEY (`PreCourse_ID`)
    REFERENCES `Tinyhub`.`Course` (`C_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)

-- -----------------------------------------------------
-- Table `Tinyhub`.`Exam`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Tinyhub`.`Exam` (
  `Exam_No` INT NULL,
  `Grades` VARCHAR(10) NULL,
  `CID` VARCHAR(10) NOT NULL,
  `No_of_Problems` INT NULL,
  `Grades_for_Problems` DECIMAL(4) NULL,
  INDEX `CID_idx` (`CID` ASC) VISIBLE,
  CONSTRAINT `CID`
    FOREIGN KEY (`CID`)
    REFERENCES `Tinyhub`.`Course` (`C_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
```

Further Discussions:

Advantages and disadvantages:

The E/R diagram for TinyHub is highly integrated with our relational model. The relational model can easily be designed using tables in our E/R model. The model designed provides abstraction at a high level for the schema, so it is easy to understand the concept and may be easily implemented if the relationship between the entities and the attributes are clear.
The model may be used to visually communicate the requirement to the end user.

It does not give a clear idea about the constraints on the model as it is high level abstract view of the database. It does not depict details about the inherent data manipulations and has limited relationship representation. It is less detailed resulting in loss of information content as information is hidden in the E/R model.The model developed for managing the details of a university is in its simplified form, as per the requirements, for there are many other aspects to be considered for a real-time university database management system. The functionalities included in this model are limited to the requirements whereas a good integrated database system would be optimized with all the domains considered. The system does not allow a program to belong to multiple departments which restricts the student from pursuing majors in a different department.