# SimpleBoun Registration

Project 3, CMPE 321, Introduction to Database Systems, Spring 2022

Due: **April 29, Friday, 23:59**

**Ask the project related questions in the Moodle forum!**

## 1 Introduction

As university students, you have all used some university registration systems. These systems allow the instructors and students to manage course registration tasks with minimal effort. Without such systems, we would need to handle everything manually, which clearly would be infeasible. On the other hand, if such a system is erroneous, it might bring more harm than benefit. Thus, a well-organized registration system is vital to ensure the validity and consistency of the actions and data.

There may be many constraints in these systems. For instance, courses might have prerequisites and students might not be allowed to take a course before passing its prerequisite courses. Another example constraint relates to the allocation of classrooms: two lectures may not be held in the same classroom simultaneously. As you can see, if the system fails to check the satisfaction of constraints, the results could be chaotic.

A university registration system is typically concerned with the instructors, students, courses, departments, grades, classrooms etc. There are many logical relationships among the elements of these systems. For instance, an instructor teaches some courses and works in a department; students take courses and are graded by the instructor. As a result, there is a great deal of information in such a system. A database is needed to store and manage the data safely and efficiently.

To understand university registration systems better, you can check Bogazici University Registration System and think about its structure, rules and constraints.

## 2 Project Description

In the previous project, you performed conceptual database design, drew ER diagrams, converted these ER diagrams into relational tables, and performed schema refinement for *SimpleBounDB*. In this project, you will go one step further and implement a course registration database with a web-based user interface (UI) using the structure you have designed in Project 2 or its improved version. *SimpleBounDB* should contain the following information:

1. **User** includes the following attributes; username, password, name, surname, email, Department ID. Each user has a unique username and is associated with exactly one department. In addition, each user is either a student or an instructor. There can be an unlimited number of users. Passwords should be encrypted using the SHA256 algorithm and stored in the database accordingly.

   (a) **Students** additionally have Student ID, a list of added courses for the current semester, completed credits and GPA attributes. A student's completed credits is equal to the sum of the credits of the courses that the student has previously been graded. Student ID must be unique. GPA is calculated as follows:

   $$\frac{\sum_{course}(course.credit * course.grade)}{total\_completed\_credits}$$

   (b) **Instructors** additionally have title attribute. Allowed titles are Assistant Professor, Associate Professor and Professor.

2. **Department** includes the following attributes: Department ID, name. Both name and id must be unique.

3. **Course** includes Course ID, name, Department ID, credits, instructor username, Classroom ID, campus, Classroom capacity, time slot, quota and a list of prerequisites. Course ID must be unique.

   - Each course belongs to the department of its instructor. No two courses can overlap in terms of classroom and time slot.

- In SimpleBoun, there are only 2 time slots for each weekday so there are 10 time slots in total. The time slots are represented with integers from 1 to 10 (e.g., First half of Monday corresponds to 1 and second half of Tuesday corresponds to 4). A course only occupies a single time slot.
- A prerequisite is a course that the student must pass before taking the other course. The Course ID of a prerequisite must be less than the ID of the succeeding course (e.g. MATH101 < MATH102). You may use string comparison to check this.
- Each Classroom ID corresponds to one physical location. Hence, campus and classroom capacity depends on the Classroom ID.

4. **Grades** have following attributes: Student ID, Course ID and grade. A pair of Student ID and Course ID uniquely identifies a grade. Grades will be given as floats (e.g. 3.5). You can assume there is no fail grade in SimpleBoun. That is, if a student has a grade from a course, he/she has passed the course.

5. **Database manager** consists of the following attributes: username and password. There exists only one database manager with a certain username. There can be at most 4 database managers registered to the system. Passwords should be encrypted using the SHA256 algorithm and stored in the database accordingly.

Three types of people will be using *SimpleBounDB*: students, instructors and database managers. All types should be able to **log in** to the system, thus you should implement an authentication mechanism. You do not need to provide sign-up, change password, and reset password options. There should be a different log in section for each role (student, instructor or database manager). After login, users will be able to perform only the operations that are defined for their roles.

# 3 Requirements

Your UI must support the following operations:

1. Database managers shall be able to log in to the system with their credentials (username and password).
2. Database managers shall be able to add new Users (Students or Instructors) to the system.
3. Database managers shall be able to delete students by providing Student ID. When a student is deleted, all personal data regarding that student must be deleted including previous grades and currently added courses.
4. Database managers shall be able to update titles of the instructors by providing instructor username and title.
5. Database managers shall be able to view all students in ascending order of completed credits. The list must include the following attributes: username, name, surname, email, department, completed credits and GPA.
6. Database managers shall be able to view all instructors. The list must include the following attributes: username, name, surname, email, department, title.
7. Database managers shall be able to view all grades of a specific student by providing Student ID. The list must include the following information: Course ID, course name, grade.
8. Database managers shall be able to view all courses of a specific instructor by providing instructor username. The list must include the following attributes: Course ID, course name, classroom ID, campus, time slot.
9. Database managers shall be able to view the grade average of a course by providing Course ID. The list must include the following attributes: Course ID, course name, grade average.
10. Instructors shall be able to log in to the system with their credentials (username and password).
11. Instructors shall be able to list all of the classrooms available for a given slot. The list must include the following attributes: Classroom ID, campus, classroom capacity.

12. Instructors shall be able to add courses by providing Course ID, name, credits, Classroom ID, time slot and quota. The Department ID of the course should be the same as the instructor's department. A course quota may not be greater than the capacity of the classroom.

13. Instructors shall be able to add a prerequisite to a course by providing its Course ID and the Course ID of the prerequisite.

14. Instructors shall be able to view all courses that they give in ascending order of Course IDs. The list must include the following attributes: Course ID, course name, classroom ID, time slot, quota and prerequisite list. Prerequisite list must be a string in the form "$prerequisite_1, prerequisite_2, ...$"

15. Instructors shall be able to view all students who added a specific course given by themselves. To view this information, the instructor should provide a Course ID. The list must include the following attributes: username, Student ID, email, name and surname.

16. Instructors shall be able update the name of a course given by themselves by providing a Course ID and course name.

17. Instructors shall be able to give grades to a student if he/she has added their courses. The instructor should provide Course ID, Student ID and grade for this operation. The grade should be stored. We will not test the case of an instructor updating a previous grade. You can assume the instructors will only enter new grades.

18. Students shall be able to list all the given courses. The list must include the following attributes: Course ID, course name, instructor surname, department, credits, classroom ID, time slot, quota and prerequisite list. Prerequisite list must be a string in the form "$prerequisite_1, prerequisite_2, ...$"

19. Students shall be able to add a course by providing a course ID. There are several constraints:

    (a) Students cannot take a course twice, that is, if a student has received a grade from the course, she/he cannot add that course.

    (b) To take the course, student must have a grade from all of its prerequisites. We will not test the case of an instructor adding a prerequisite for a course after a student has added the course.

    (c) Students cannot add a course if the quota is full. At the beginning, added courses of students will be provided. The quota for a course is the maximum number of students who can take that course.

20. Students shall be able to view the courses that they're currently taking and have taken previously. The list must include the following attributes: Course ID, course name and grade. If the student is taking the course this semester, the grade field will be null.

21. Students shall be able to search a keyword and view the courses that contain this keyword in their names. The list must include the following attributes: Course ID, course name, instructor surname, department, credits, classroom ID, time slot, quota and prerequisite list. Prerequisite list must be a string in the form "$prerequisite_1, prerequisite_2, ...$"

22. Students shall be able to filter courses of a specific department with respect to its campus, and an inclusive range between minimum and maximum credits (e.g., courses belonging to Computer Engineering Department, given at the North Campus, with a minimum of 3 and a maximum of 5 credits). This must be implemented as a **stored procedure**. Parameters of this procedure are department ID, campus, minimum credits, and maximum credits.

23. You **must** handle the following cases with **triggers**:

    (a) When an instructor grades a student from a course, the completed credits of the student will be incremented by the credits of that course. When an instructor grades a student, the student's GPA must be updated.

    (b) When an instructor adds a course, the quota must not exceed the classroom capacity. If that's not the case, the system must prevent the operation.

    You can write more triggers if it is needed.

# 4 Notes

- In the scope of this project, we are more interested in the functionality of the web interface, rather than the styling of UI. In other words, it is totally fine to create a system that satisfies all requirements and has zero line of CSS and Javascript.
- The allowed languages are PHP, Java, JavaScript, and Python. You can use a framework, however, you **must** write the SQL queries and boot the database server yourself. Note that you should set up the database and create the tables on your own. You are **not** allowed to use any tool that helps with these parts such as ORM(Object Relational Mapping).
- Also, do **not** use database connector as your query supplier. Use it only as a query executor by running it with SQL strings written by you.
- You are not expected to deploy your system. So, it is fine if it runs in your local.
- You are free to use **relational database** server of your choice. However, we will **not** accept submissions with non-relational databases or no database servers. Also, SQLite is **not** applicable for this project.

# 5 Report & Grading

This project can be implemented either individually or as a team of two people. You are free to change teams in the upcoming projects. The submission must include your **code**, your updated ER diagrams, and a READ.ME file that describes how to run your code. Place all the required files into a folder named with the student IDs of the team members separated by an underscore (e.g., `2017400200_2018700120`). Zip the folder for submission and name the `.zip` file with the same name. Submit the `.zip` file through Moodle until the deadline. The system will be evaluated during a demo session that will be arranged. Demo day(s) will be announced. Before the demo date, we will share the data that will be used during the demo in Excel format. You **must** add the entries before the demo. **Any other submission method is not allowed**. Each group should submit **one** .zip file.

# 6 Late Policy

You are allowed a total of 5 late days on the projects with no late penalties applied. You can use these 5 days as you wish. For example, you can submit the second project 2 days late, then the third project 3 days late. In that case, you will have to submit the fourth project on time. No late submissions will be accepted after you use these 5 extra days. If you change your team, the team's late days used so far will be the maximum number of late days used by any of the team members.