**A logo of a college

Description automatically generated**

**Texas A&M University**

**Database System Course Project1**

**University Management System**

**Name: Zizhan Xiong**

**UIN: 534002298**

Part A

A1. Project Description:

My system is a university management system. This website uses Python/Flask as backend framework and PostgreSQL database. The data is fabricating data generated by me using python. The system mainly contains five different services. The first one is adding a new student. By entering the name, UIN, birth date, gender of a student, we can add this data to existing database. The second one is Search score. By entering the UIN of a student and course id the student taken, it will show the score of the student in this course. The third one is searching for a professor for a course. By entering the course id, it will show all the information of the professor taking this course. It will allow students to conveniently choose a professor when registering for a course. The fourth one is adding a new professor. By entering the information of a professor, including professor UIN, professor name, department, age and gender, we can add this new professor into the database. The fifth one is searching for course information. By entering the course id, students can find all the information about this course.

This system includes three entity sets and two relationship sets.

1. Student has UIN, Student name, Birth year, Gender, Phone and Major

2. Professor has Professor UIN, Professor name, Department name, Age and Gender

3. Course has Course id, credit, CRN, Department and Start Date

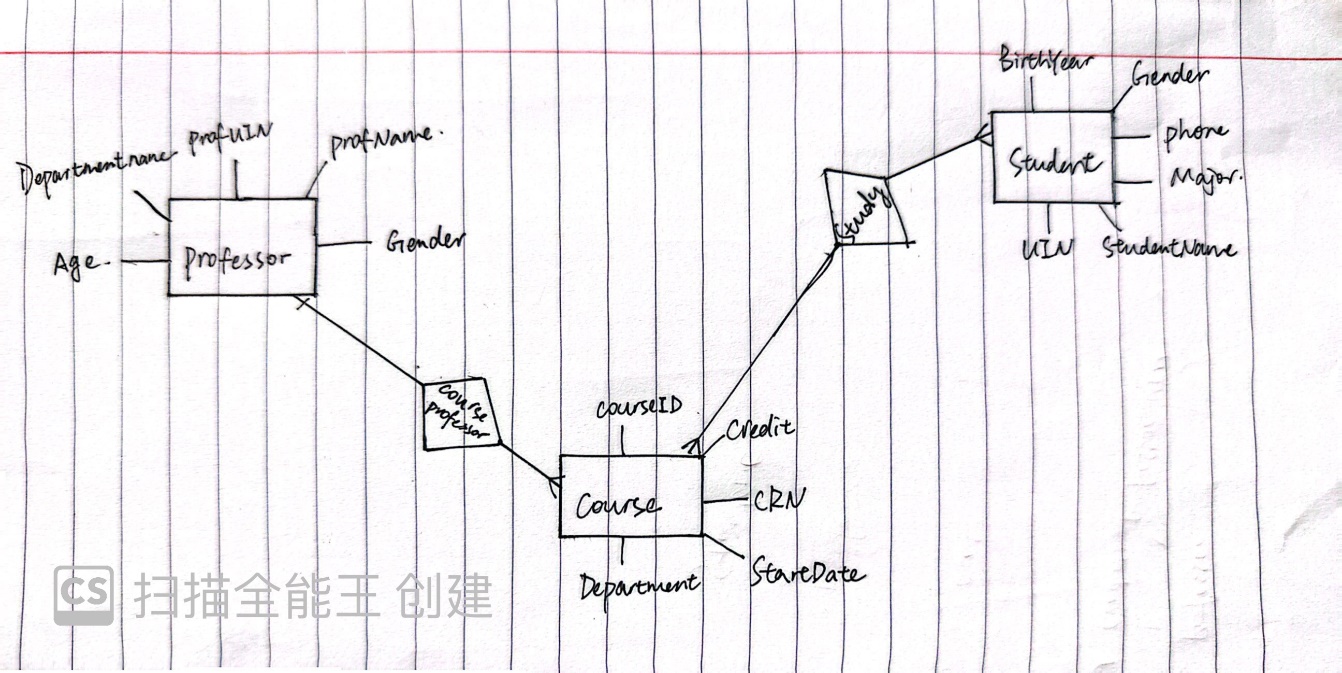
There is a course professor relationship between professor and course. This relation simply shows which course is taught by which professor. This is a one-many relationships. Since a course can only be taught by one professor and a professor can take many courses. In this table, it contains the course id and professor UIN.

There is also a study relationship between student and course. This relationship simply shows which course is taken by which student. This is a many-many relationships. Since a student can take many courses and a course can have many students. In this table it contains the course id, student UIN and score of this student in this course.

The data I used is generated by myself in python. I generally generate 2000 different courses and 10000 different professors and 20000 different students to form these tables and relationships.

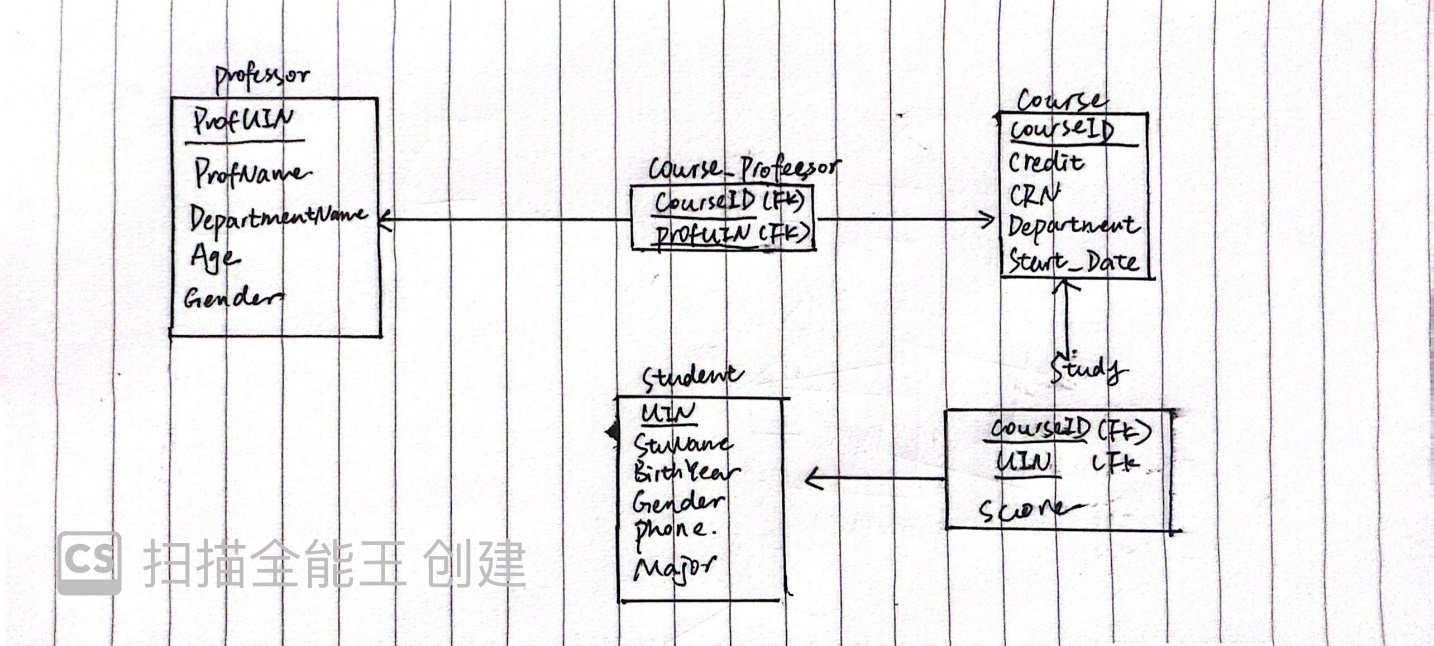
The reason why I chose to do this system is because I have used it for several years and I want to do a simple version copy of it on my own. In addition, since I have used it for a long time, I am familiar with the relationship between different sets and what kind of sets it should contain. I think it is a good start for my first project. Not such difficult and meaningful.

A2. ER diagram.



Part B

B1. schema



B2. Functional Dependency

1. CREATE TABLE course\_professor(

CourseID VARCHAR(10),

Credit INT,

ProfUIN INT,

PRIMARY KEY (CourseID)

);

ALTER TABLE Course\_Professor DROP COLUMN Credit;

In the Course table, the nontrivial FD is:

Course ID🡪 ProfUIN

Notice that I have dropped Credit and this column is moved to Course table.

This relation is already in BCNF.

There is no MVD in this relation and is already in 4NF.

2. CREATE TABLE professor(

ProfUIN INT,

ProfName VARCHAR(30),

DepartmentName VARCHAR(30),

Age INT,

Gender VARCHAR(6),

PRIMARY KEY (ProfUIN)

);

In the professor table, the nontrivial FD is:

ProfUIN 🡪 ProfName, DepartmentName, Age, Gender.

This relation is already in BCNF.

There is no MVD in this relation and is already in 4NF.

3. CREATE TABLE student(

UIN INT,

StuName VARCHAR(30),

BirthYear INT,

Gender VARCHAR(6),

Phone FLOAT,

Major VARCHAR(20),

PRIMARY KEY(UIN)

);

In the student table, the nontrivial FD is:

UIN 🡪 StuName, BirthYear, Gender, Phone, Major

This relation is already in BCNF.

There is no MVD in this relation and is already in 4NF.

4. CREATE TABLE study (

CourseID VARCHAR(10),

UIN int,

Score INT DEFAULT NULL,

PRIMARY KEY (CourseID,UIN)

);

In the study table, the nontrivial FD is:

CoursrID, UIN 🡪 score

This relation is already in BCNF.

There is no MVD in this relation and is already in 4NF.

5. CREATE TABLE course(

CourseID VARCHAR(10),

Credit INT,

CRN INT,

Department VARCHAR(30),

Start\_Date FLOAT,

PRIMARY KEY(CourseID)

);

In the course table, the nontrivial FD is:

CourseID 🡪 Credit, CRN, Department, Start\_Date

This relation is already in BCNF.

There is no MVD in this relation and is already in 4NF.

The SQL code is in an SQL file called Project 1.

Part C

C1. Data Generation

**Data Generation for Course\_professor Table**

Objective:

The goal of this script is to create and populate a course\_professor CSV file with simulated data for 2000 course entries. The data generated is intended to represent a variety of courses associated with respective professors at a university.

Methodology: The script employs Python's random module to generate randomized data for three fields within the course\_professor table: CourseID, Credit, and ProfUIN.

Procedure:

1. CourseID Generation:

CourseID is composed of a department prefix and a course number. The department prefix is randomly selected from a predefined list of department codes such as 'CSCE', 'ENG', 'MATH', etc. The course number is a random integer between 100 and 699, representing a plausible course level within the university structure. The concatenation of department prefix and course number forms the CourseID. To ensure uniqueness, each CourseID is checked against a set of existing IDs before it's added to the list.

Credit Assignment:

Each course is assigned a random credit value between 2 and 4, which aligns with typical university credit hours for a course.

Professor UIN Assignment:

Each course is assigned a ProfUIN, which is a randomly generated integer between 0000 and 9999, presumably representing a unique identifier for a professor.

CSV File Creation:

The script creates a CSV file named course\_professor.csv on the desktop of the user.

It employs Python’s csv module to write the headers and the generated rows into the file. The CSV file is structured with the headers CourseID, Credit, and ProfUIN.

Data Population:

A loop runs to generate 2000 rows of data, calling upon the aforementioned functions to populate each field. Each row represents a unique course offering and the professor assigned to it.

Output: The script generates a CSV file, course\_professor.csv, which is stored at the specified desktop path. Each entry in the CSV file contains unique and randomly generated data for the CourseID, Credit, and ProfUIN fields, resulting in a comprehensive dataset for 2000 courses.

**Data Generation for Professor Table**

Objective:  
The script's purpose is to generate a simulated dataset for a professor table. The dataset includes unique identifiers (UINs), names, department affiliations, ages, and genders for a set of university professors.

Methodology:  
Using Python's names and random libraries, alongside pandas for data manipulation and CSV generation, the script creates a realistic and randomized list of professors.

Procedure:

Professor Data Generation:

ProfUIN is generated in sequence starting from 0, acting as a unique identifier. ProfName is obtained using the names.get\_full\_name() function, which produces a realistic full name. DepartmentName is selected randomly from a predefined list of department codes. Age is randomly chosen within a range of 25 to 70 years, assuming this represents the typical working age range for professors. Gender is randomly assigned with equal probability of being 'Male' or 'Female'.

CSV File Creation:

The script uses pandas.DataFrame to structure the data into a tabular format. It then writes this DataFrame to a CSV file named professor.csv, saved to the user's desktop path.

Data Population:

The process is repeated for num\_professors iterations, specified as 10,000 entries for this dataset.

Output:  
A CSV file titled professor.csv is generated and stored at the specified location. The file includes columns for ProfUIN, ProfName, DepartmentName, Age, and Gender, containing data for 10,000 professor entries.

**Data Generation for Student Table**

Objective:  
The aim of this script is to generate a dataset for a Student table, simulating the student records for a university. The dataset includes a range of information, including unique identifiers (UINs), student names, birth years, genders, phone numbers, and majors.

Methodology:  
The script uses Python's built-in modules, csv for CSV file operations, random for generating random numbers and selecting random choices, and names for generating realistic full names.

Procedure:

Student Data Generation:

UIN is incremented starting from 10001 to ensure uniqueness. StuName is generated using the names.get\_full\_name() method, which gives a realistic full name. BirthYear is randomly selected between 1999 and 2005, reflecting an age range suitable for undergraduate students. Gender is randomly chosen between 'Male' and 'Female'. Phone is generated by the generate\_phone() function, creating a pseudo phone number and converting it to a float, formatted as x.yyyzzzzzzz. Major is randomly selected from a list of common academic disciplines.

CSV File Creation:

The script initiates a CSV file with the specified headers. Each student's data is written as a row into the Student.csv file saved to the user's desktop.

Data Population:

The loop generates data for 20,000 students, populating the student records with the generated data.

Output:  
A CSV file named Student.csv is created, holding 20,000 entries with data fields for each student, corresponding to the columns UIN, StuName, BirthYear, Gender, Phone, and Major.

**Data Generation for Course Table**

Objective:  
The purpose of this script is to generate a dataset for the Course table, which includes information on various courses offered at a university. The dataset comprises course identifiers, credit values, course registration numbers (CRNs), department names, and start dates.

Methodology:  
The script uses pandas for reading an existing course\_professor.csv file and for data frame manipulation, as well as Python's standard library for randomization and regex operations.

Procedure:

Data Retrieval: The script reads an existing CSV file, course\_professor.csv, to obtain a list of unique CourseIDs.

Course Data Generation:

For each unique CourseID, the following attributes are generated:

Credit: Randomly assigned as an integer between 2 and 4.

CRN: A unique course registration number starting from 10000 and incrementing by 1 for each course.

Department: Extracted from the CourseID using regex to match non-digit characters.

Start\_Date: Randomly chosen from a list of two possible values, 1.16 and 3.22, presumably representing the start of the semester.

Data Population:

The generated data for each course is appended to a list.

CSV File Creation:

A new panda DataFrame is created from the list of course data.

The DataFrame is then written to a CSV file named Course.csv, saved to the user's desktop within a subfolder named Table generation.

Output:  
A CSV file entitled Course.csv is generated, including detailed course information for each entry. The file is structured with the headers CourseID, Credit, CRN, Department, and Start\_Date.

**Data Generation for Study Table**

Objective:  
The script's aim is to simulate a dataset for a Study table, representing student enrollments in courses along with their respective scores.

Methodology:  
Employing pandas, random, and itertools, the script creates associations between students and courses by assigning random course IDs to student UINs and generating random scores for these courses.

Procedure:

Data Extraction:

The script reads from two CSV files, course\_professor.csv and Student.csv, which contain lists of course IDs and student UINs, respectively.

Course and Student Associations:

It uses itertools.cycle to create an iterator that cycles through the course IDs, allowing for the assignment of each course to multiple students.

Each student is assigned a course ID from this cycling list, ensuring a broad and even distribution of course enrollments.

Score Assignment:

A random score between 0 and 100 is generated for each student-course pairing, signifying the student's performance in the course.

Data Aggregation:

The script aggregates the course IDs, student UINs, and scores into a list of dictionaries, each representing a record in the Study table.

CSV File Creation:

A new pandas DataFrame is created from the aggregated data.

This DataFrame is then saved to a CSV file named Study.csv in the Table generation subfolder on the user's desktop.

Output:  
The resulting Study.csv file is structured with columns for CourseID, UIN, and Score, reflecting the enrollment and performance data for students across various courses.

C2. Test

After generating these tables, I import them into my database in Pgadmin4. Then I write some SQL code to test my tables.

**Query to Find Students Enrolled in Biology Courses**: This query selects student identifiers (UINs), student names (StuNames), course identifiers (CourseIDs), and scores from the study table. It joins the study table with the course table to get course details and the student table to get student details. The WHERE clause filters the results to only include courses whose IDs start with 'BIO' followed by two characters (designated by the underscore placeholders). This typically represents students enrolled in Biology courses.

**Query to Find Students with Scores Below 60**: This query retrieves the same columns as the first one, but this time, it filters the results to show only the instances where a student's score in a course is below 60. This could be used to identify students who are failing or who may need additional help in their courses.

**Query to Find Senior Professors and Their Courses**: This query selects ages and names of professors, their department names, and the courses they teach from the Course\_Professor table. It joins with the professor table to match professors to the courses they're teaching. The WHERE clause filters to show only those professors who are 50 years old or older. This might be used to gather data for analyzing the distribution of senior faculty across departments or planning for upcoming retirements.

The tables generated by these three are stored in test table folder.

Part D

Introduction

The University Management System is designed to facilitate the administrative, academic, and management tasks of a university. It serves as a central platform for managing students, professors, courses, and more, providing an efficient and user-friendly interface for both administrators and users.

System User Interface

The system's user interface is web-based, developed using Flask, a Python web framework. The interface is designed to be intuitive and accessible, ensuring users can easily navigate through the various functionalities offered by the system. Key features of the user interface include:

Home Page: A welcoming entry point that provides quick access to all system functionalities.

Forms: Dedicated pages for adding new students and professors, equipped with form fields for inputting relevant data.

Search Pages: Custom search interfaces that allow users to search for course scores, professor information, and course details based on specific criteria.

Results Display: Pages that neatly display search results, providing detailed information in a clear and structured format.

Functions Offered by the System

The system offers a range of functions to facilitate university management tasks:

Add Student: Allows administrators to add new student records, including UIN, name, birth year, gender, phone number, and major.

Search Scores: Enables users to search for students' scores by entering a student's UIN and a course ID.

Search Professor of a Course: Users can search for professor details associated with a specific course ID.

Add Professor: Facilitates the addition of new professor records, capturing details such as professor UIN, name, department, age, and gender.

Search Course Information: Provides the ability to search for detailed information on courses by entering the course ID.

Implementation in SQL

The system's functions are supported by a backend database, with SQL queries enabling the retrieval and manipulation of data:

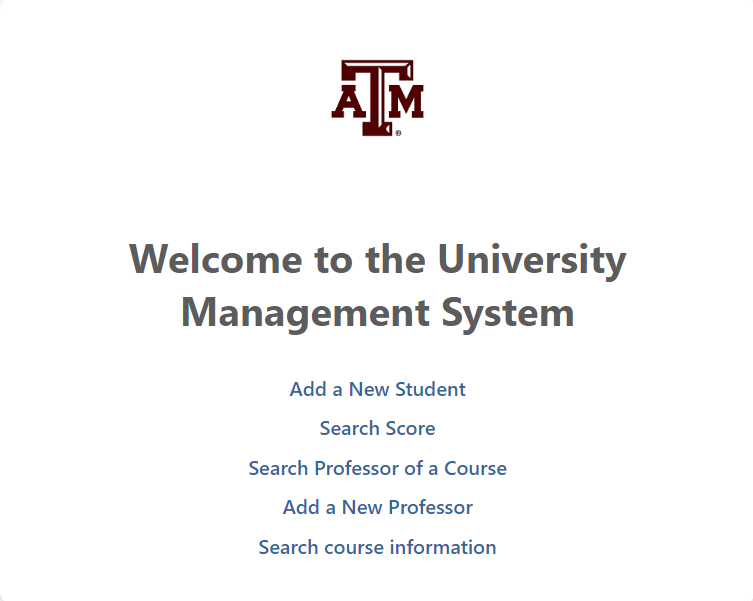
Add Student/Professor: These functions involve SQL INSERT statements that add new records to the respective student or professor tables.

Search Scores: Implemented using an SQL SELECT query that joins the study, course, and student tables, filtering results based on the provided student UIN and course ID.

Search Professor of a Course: Utilizes an SQL JOIN operation between the course\_professor and professor tables, filtering by the given course ID to return the associated professor's details.

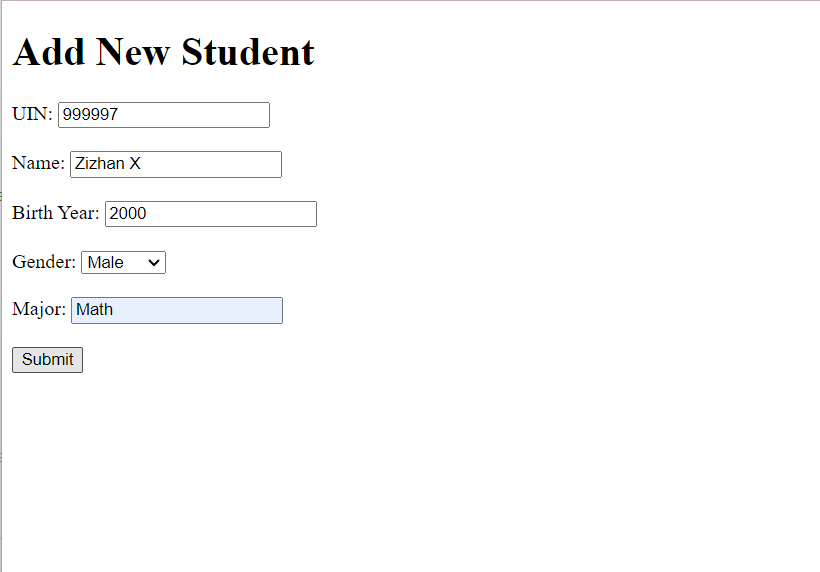
Search Course Information: This function executes an SQL SELECT query on the course table, filtering by the course ID to retrieve detailed course information.

Below is how user may use my interface

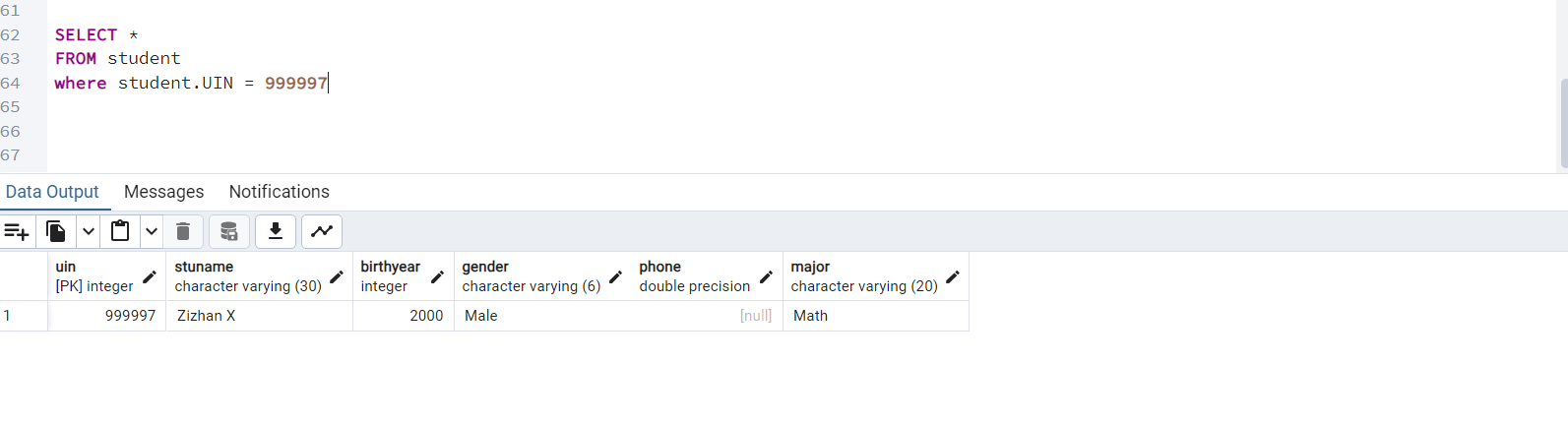


This is the home page.

Add a new Student

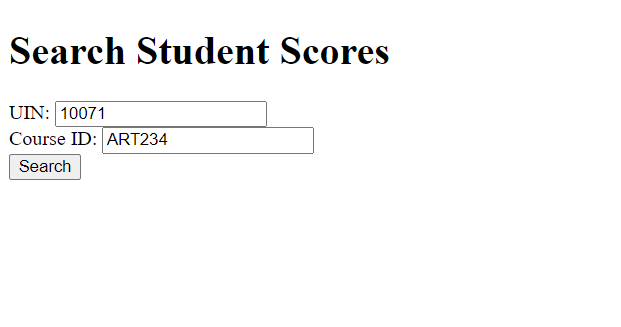


By typing this information, the information of this student will be added to our database.

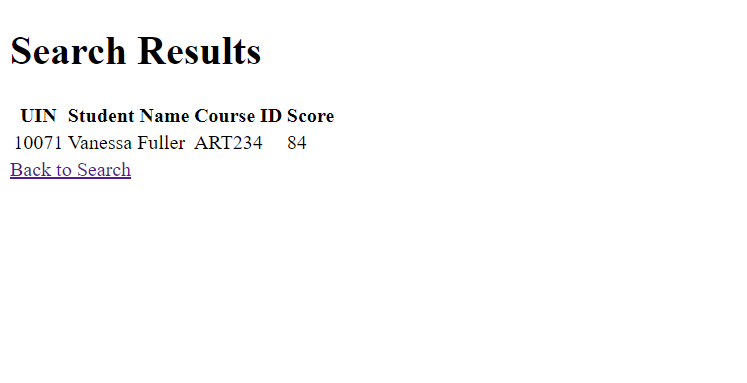


Search score

By entering the UIN of the student and course id, it will show the score of the student in this couse.



Below is the search results.

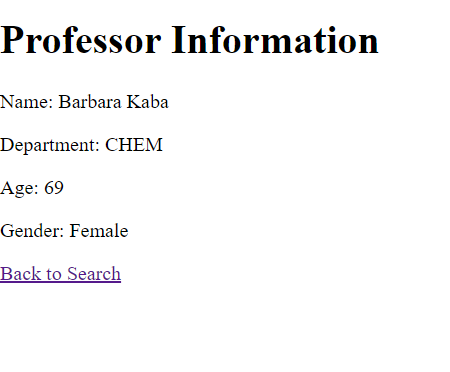


Search Professor of a course

By entering the course id, the system will show all information of the professor teaching this course.

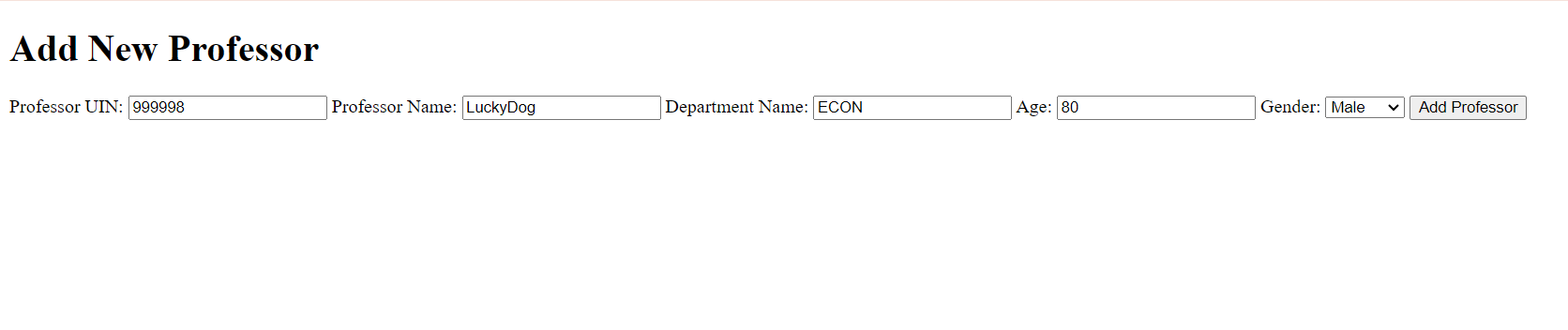
A search box with black text

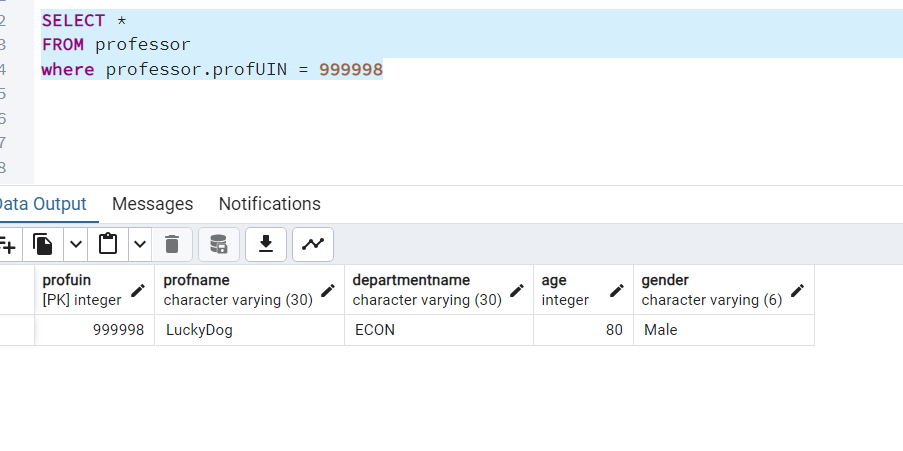
Description automatically generated



Add a new professor

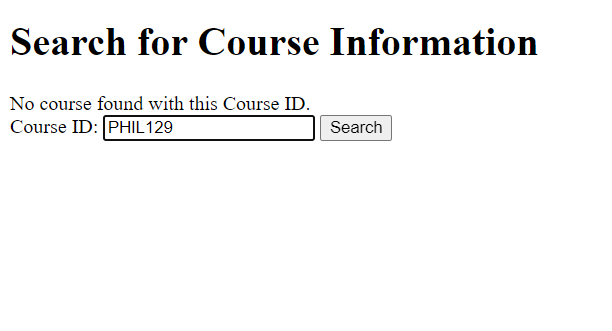
By entering the information of the professor, the system will add it to the database

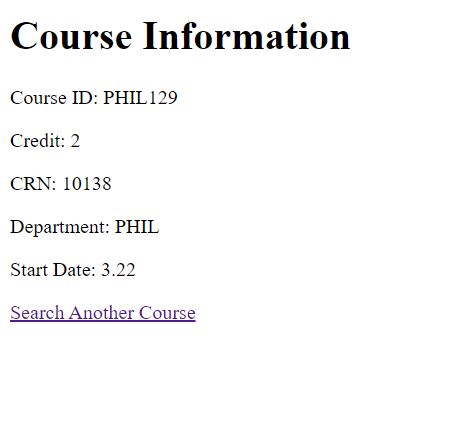




Seacrh course information

By entering the course id, it will show all information of the course.





Part E

Discussion:

Developing a database application, particularly for managing university operations, is a complex yet rewarding endeavor. Throughout the process, various challenges arise, but overcoming them significantly enhances both technical skills and problem-solving abilities. The first difficulty that I have met is to generate such big data on my own and make sure the key is unique. After several times debugging, I finally generated the table with unique key. The second difficulty is to import the data generated into the database in Pgadmin4. For the initial several times, I could not make the column match the table create in the SQL. After some adjustments, I successfully import them. Finally, the biggest challenge is to use flask and frontend languages to build my interface. I didn’t learn this part before. So I have to learn it on my own. The function in the interface may seem to be simple but I have try my best.

The most important thing that I have learned during this project is I have experienced to build a interface with frontend languages which I never get in touch with before. Having this experience, in the future, I can build a more complex interface with more functions and services in it.