**Course**: COP 4710: Database Management Systems (Any Inquiries regarding grading of the assignment should be communicated firstly with Ms. Kammali and then Mr. Ruiz. If your issue is still unsolved, you should communicate with Dr. Bahreini). Any inquiries regarding the clarity of this document should be communicated with Dr. Bahreini.

**Professor and instructor of the course**: Kiavash Bahreini, Ph.D. (he/him/his) (kiavash.bahreini@fiu.edu)

**Teaching assistant and grader**: Ricardo Ruiz, MSc. (he/him/his) (rruiz101@fiu.edu)

**Grader:** Jhansi Lakshmi Kammili (she/her) (jkammili@fiu.edu)

**Department**: Knight Foundation School of Computing and Information Sciences

**University**: Florida International University

**Semester**: Summer 2022-23

**Project:** Student Registrations System at ABC University

Disclaimer: This document has been created for internal usage at Florida International University (FIU), specifically for the course COP 4710: Database Management Systems students. Please do not share or distribute it with third parties, persons, or organizations outside FIU.

**Note: The deadline to submit this project will be Saturday, July 15, 2023, at 11:58 pm. However, because you are given about two months to submit your project in advance, there will be no late submission for this project. Otherwise, if you want late submissions, I can create the deadline until July 9, 2023, at 11:58 pm and then let you submit your project with the %15 daily reduction until July 15, 2023, at 11:58 pm.**

Please complete the following information:

**Student name: Camron**

**Student surname: Cisneros**

**Student email address: ccisn008@fiu.edu**

**Student panther ID: 6187231**

**Student login ID (if different than your Panther ID):**

**Course code: COP4710 U01C 1235**

**Other information to mention (if any):**

**Student Registrations System at ABC University**

**Introduction**

In today's digital age, most universities have adopted sophisticated database systems to manage their students' information. University ABC is no exception and has developed its own Student Registrations System to keep track of student records. However, the system lacks a crucial feature that can make student registration easier - **a register and login page**. Students must visit the university's enrolment department in person to manually register their information. This approach is not only time-consuming but can also lead to errors in record-keeping.

To address this issue, the university has decided to undertake a project to develop a database system that will allow students to register and manage their information quickly. The project aims to create a user-friendly system for students to register their details, such as personal information, academic records, and extracurricular activities. The system will also provide features to allow students to view and update their records, track their academic progress, and communicate with the university staff.

The project is an essential step for the university toward digital transformation, making it easier for students to manage their academic life. The new system will improve the student experience and help the university staff streamline administrative processes, reduce errors, and enhance data security. With the successful implementation of the new system, the university can expect to see increased student satisfaction, improved administrative efficiency, and a better overall academic environment.

The project will involve creating a Conceptual E-R model and a Relational model for the database, using Structured Query Language in MySQL database, creating a simple GUI for only a register and login page of an application using a programming language to connect to the MySQL database, and performing select, insert, update, and delete statements through GUI.

**Entities**

It would help if you considered defining **at least six entities**. For example, the following six entities **may be** required for the database system:

**Student** - This entity represents a student and includes attributes such as student ID, name, address, phone number, email address, and date of birth.

Example:

| Student ID | First Name | Last Name | Address | Phone Number | Email | Date of Birth |

|---------------|----------------|----------------|------------|---------------------|---------|-------------------|

| 1 | John | Doe | 123 Main St. | (555) 555-1234 | johndoe@email.com | 01/01/2000 |

**Course** - This entity represents a course and includes attributes such as course ID, course name, instructor name, start time, end time, and room number.

Example:

| Course ID | Course Name | Instructor Name | Start Time | End Time | Room Number |

|--------------|-------------------|------------------------|---------------|--------------|---------------------|

| 1 | Introduction to Computer Science | Dr. Smith | 9:00am | 10:30am | Room 101 |

**Enrollment** - This entity represents a student's enrolment in a course and includes attributes such as enrolment ID, student ID, course ID, and grade.

Example:

| Enrolment ID | Student ID | Course ID | Grade |

|-------------------|---------------|---------------|---------|

| 1 | 1 | 1 | A |

**Schedule** - This entity represents a student's schedule and includes attributes such as student ID, course ID, start time, end time, and room number.

Example:

| Student ID | Course ID | Start Time | End Time | Room Number |

|---------------|---------------|---------------|--------------|--------------------|

| 1 | 1 | 9:00am | 10:30am | Room 101 |

**Department** - This entity represents a department and includes attributes such as department ID and name.

Example:

| Department ID | Department Name |

|---------------------|--------------------------|

| 1 | Computer Science |

**Professor** - This entity represents a professor and includes attributes such as professor ID, name, address, phone number, and email address.

Example:

| Professor ID | First Name | Last Name | Address | Phone Number | Email |

|-----------------|----------------|----------------|------------|---------------------|---------|

| 1 | Alex | Doe | 369 Sun St. | (888) 333-6699 | alexdoe@email.com |

**Task 1**: Create a Conceptual E-R Model for the Database

The primary objective of the first task of this project is to develop a conceptual E-R model for the database that will define the structure of the database, relationships, and attributes. To create the E-R model, at least six entities, such as students, faculty, courses, departments, grades, and extracurricular activities, must be defined along with their relationships and attributes.

These entities will be the foundation of the database and ensure that the data can be accurately captured and easily accessed. In addition, the E-R model will allow University ABC to understand the data flow within the system, ensuring that the database structure meets the organization's needs.

A tool like Lucidchart can be used to create the E-R model. The E-R model created will serve as a blueprint for designing the database, guiding the development of data storage and access capabilities for University ABC.

In conclusion, creating a conceptual E-R model is a critical first step in designing a database system that is functional and efficient for University ABC. Defining at least six entities, relationships, and attributes will help ensure the system meets the organization's needs, allowing for accurate data capture and easy access.

**Note**: several other tools are available for creating E-R models besides Lucidchart. Some of the most popular tools for creating E-R diagrams include:

Draw.io: A free and open-source diagramming tool that can create a wide range of diagrams, including E-R diagrams.

Gliffy: A cloud-based diagramming tool that allows users to create E-R and other diagrams.

SmartDraw: A diagramming tool that provides a range of templates, including E-R diagrams, to help users create professional-looking diagrams quickly.

Creately: A cloud-based diagramming tool that offers a range of templates, including E-R diagrams and collaboration features for teams.

ConceptDraw DIAGRAM: A diagramming tool that provides a range of templates and libraries, including E-R diagrams, and allows users to create custom templates.

These tools offer functionality similar to Lucidchart, each with strengths and weaknesses. The choice of tool depends on factors such as the user's preferences, budget, and the project's specific needs.

**Task 2**: Create a Relational Model for the Database in MySQL

In this task, you will create a relational model for the database in MySQL (or another DBMS of your choice). This approach is an essential step as the relational model will define the structure of the database, including the tables, columns, and relationships between them. The relational model will be based on the conceptual E-R model created in Task 1, which identifies at least six entities that will be used in the database.

To create the relational model, the project team must define the tables used in the database, along with their respective columns and data types. In addition, the relationships between the tables must also be defined, such as the primary key-foreign key relationships that link records in different tables.

For example, the project team can create a "students" table using SQL script. The "students" table may contain student\_id, first\_name, last\_name, address, phone\_number, email, and date\_of\_birth. The student\_id column will serve as the primary key for the table. Other columns, such as first\_name, last\_name, and email, will be defined as varchar data types, while columns, such as date\_of\_birth, will be defined as the date data type.

It would help if you created additional tables as the conceptual E-R model required, ensuring that each table is appropriately linked to other tables through primary key and foreign key relationships. By creating a well-defined relational model, the database can be structured easily to query and manage, allowing for efficient data storage and retrieval. Following are two pieces of SQL script to create two tables as two examples for you:

CREATE TABLE students (

student\_id INT PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

address VARCHAR(100),

phone\_number VARCHAR(20),

email VARCHAR(50),

date\_of\_birth DATE

);

CREATE TABLE courses (

course\_id INT PRIMARY KEY,

course\_name VARCHAR(50),

instructor\_name VARCHAR(50),

start\_time TIME,

end\_time TIME,

room\_number VARCHAR(20)

);

**Task 3**: Use Structured Query Language

Task 3 requires using Structured Query Language (SQL) to manage the data in the database. SQL is a standard language used for managing relational databases. You will use SQL to create at least six tables and insert, update, delete, and retrieve data from the database.

In this task, you will focus on performing operations related to the **registrations and login of a student**. Therefore, you must write SQL statements such as select, insert, update, and delete for the Student table. In addition, you should write at least six SQL queries to retrieve information from the Student table based on specific criteria. For instance, you might want to retrieve the names of all students in the table or retrieve a student's password based on their ID.

In addition to retrieving data, you will write one SQL statement to insert data into the Student table, one SQL query to update data in the Student table based on the student's ID, and one SQL query to delete a student based on their ID.

By writing SQL queries, you will show how to retrieve, manipulate, and manage data in the database. This approach will help you better understand how the database system works and how data can be organized, queried, and managed efficiently.

**Task 4**: Create a Simple GUI for an Application

This task is focused on creating a simple graphical user interface (GUI), a web page, a command prompt, or a terminal, based interface for an application that can interact with the MySQL database created in Task 3. For example, the GUI/web/command prompt/terminal application will allow users to select, insert, update, and delete statements on the database. In addition, the students must implement the application's register and login page **using any programming language** like Java or Python. For example, JDBC (Java Database Connectivity) can connect the Java application to the MySQL database. In contrast, PyMySQL can be used in Python to connect to the MySQL database.

The GUI/web/command prompt/terminal application will be designed to make it easy for users to interact with the database using buttons (input values), text fields (output values), and other user interface components (input/output values). The students can use any GUI development tools, such as JavaFX or Swing in Java, to create the interface. In Python, the Tkinter library can create a graphical interface to communicate with the MySQL database. If you do not know GUI, you will use the command prompt on Windows machines or the terminal on macOS machines to develop your application. The following YouTube videos are two remedies for command prompt-based implementation of this project:

1. [Java Collection - CRUD Operation - INSERT, UPDATE, DELETE, SEARCH, and DISPLAY of Employee Collection - YouTube](https://www.youtube.com/watch?v=O-XrUJj83E0)

2. [Java JDBC CRUD Tutorial (SQL Insert, Select, Update, and Delete Examples) - YouTube](https://www.youtube.com/watch?v=1pA64-foiz8)

Once the GUI or other interfaces is created, users can register by providing their details, which will be inserted into the MySQL database using the insert statement. Users can then log in using their registered credentials, and the GUI should authenticate them by querying the MySQL database using the select statement. Users should also be able to update their details and delete their accounts using the update and delete statements.

Overall, Task 4 requires you to create a functional GUI that interacts with the MySQL database created in Task 3, enabling users to perform basic CRUD (Create, Read, Update, Delete) operations on the database.

**Note**: Here are a few more programming language examples and the corresponding database connections that can be used for Task 4:

C# - You can use the .NET Framework's ADO.NET library to connect to a MySQL database. The connection string looks something like this:

string connString = "SERVER=localhost;DATABASE=myDatabase;UID=myUsername;PASSWORD=myPassword;";

PHP - PHP has built-in support for MySQL databases through the MySQLi and PDO extensions. Here is an example connection string using MySQLi:

$conn = mysqli\_connect("localhost", "myUsername", "myPassword", "myDatabase");

Ruby - You can use the Ruby MySQL library to connect to a MySQL database. Here is an example connection code:

require 'mysql2'

client = Mysql2::Client.new(

host: 'localhost',

username: 'myUsername',

password: 'myPassword',

database: 'myDatabase'

)

Node.js - You can use the mysql2 library in Node.js to connect to a MySQL database. Here is an example code:

const MySQL = require('mysql2/promise');

const pool = mysql.createPool({

host: 'localhost',

user: 'myUsername',

password: 'myPassword',

database: 'myDatabase'

});

These are just a few examples, and many other programming languages and libraries can be used to connect to a MySQL database.

**Task 5**: Perform Select, Insert, Update, and Delete Statements Through GUI

In this task, you must ensure that the GUI you created in task 4 can perform the necessary SQL statements for registration and login scenarios. This approach means application users can interact with the database using the GUI interface to perform select, insert, update, and delete operations.

For example, a user might use the GUI to input their registration details, such as their name, email address, and password. The application will then perform an insert statement on the database to add the user's details to the relevant table. Similarly, users might use the GUI to input their login details, such as their email address and password. The application will then perform a select statement on the database to check if the user's details match those in the relevant table and, if so, allow the user to log in.

You will need to ensure that the necessary SQL statements are written and implemented in the backend code of the application, allowing the GUI to communicate with the database and perform the desired operations. This approach may involve using libraries and frameworks specific to your chosen programming language to create the application and implementing error handling and validation to ensure data integrity and prevent security vulnerabilities.

**Conclusion**:

In conclusion, developing a user-friendly database system that allows ABC University students to register and manage their information efficiently is crucial for University ABC in today's digital age. By implementing this project, the university is taking a significant step towards improving the overall academic experience for its students. The new system will make it easier for students to register and manage their information and help the university staff streamline administrative processes, reduce errors, and enhance data security. With the successful implementation of this project, the university can expect to see increased student satisfaction, improved administrative efficiency, and a better overall academic environment. This project represents an essential investment in the university's future and will positively impact students and staff.

**What should you deliver?**

1. Write all the entity names and their columns and other information similar to how I wrote in the **Entities** section (9 points).

**YOUR ANSWER**:

**STUDENTS**

**| Student ID | First Name | Last Name | Address | Phone Number | Email | Date of Birth |Pass|**

**COURSES**

**| Course ID | Course Name | Instructor Name | Start Time | End Time | Room Number | Department ID | Professor ID |**

**ENROLLMENT**

**| Enrolment ID | Student ID | Course ID | Grade |**

**SCHEDULES**

**| Student ID | Course ID | Start Time | End Time | Room Number |**

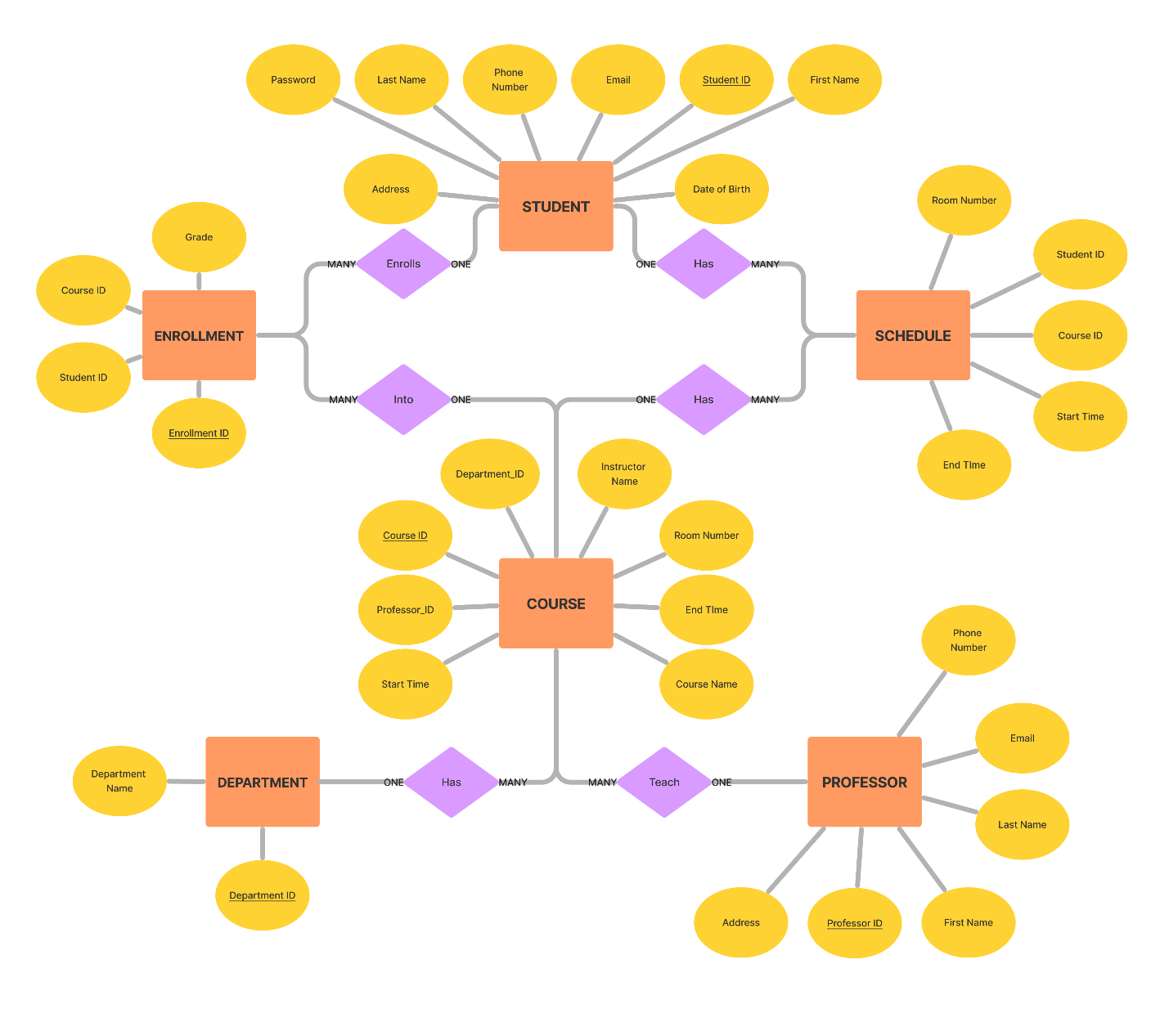
**DEPARTMENTS**

**| Department ID | Department Name |**

**PROFESSORS**

**| Professor ID | First Name | Last Name | Address | Phone Number | Email |**

2. Perform task 1, capture an image of the Lucidchart's conceptual E-R model for the database, and add it to this section. You may use any other tools for this task (9 points).

**YOUR ANSWER**: 

3. Perform task 2, add a database diagram for your tables and their relationships in MySQL (or another DBMS of your choice), and add it below. Then, add all the SQL codes (CREATE TABLE, ALTER TABLE, CREATE PRIMARY KEY, FOREIGN KEY, etc.) related to creating the relational model for the database in MySQL in this section (9 points).

**YOUR ANSWER**:

CREATE TABLE students (

  student\_id INT PRIMARY KEY,

  first\_name VARCHAR(50),

  last\_name VARCHAR(50),

  address VARCHAR(100),

  phone\_number VARCHAR(20),

  email VARCHAR(50),

  date\_of\_birth DATE

);

ALTER TABLE students

ADD pass VARCHAR(50);

CREATE TABLE courses (

  course\_id INT PRIMARY KEY,

  course\_name VARCHAR(50),

  instructor\_name VARCHAR(50),

  start\_time TIME,

  end\_time TIME,

  room\_number VARCHAR(20)

);

ALTER TABLE courses

ADD COLUMN department\_id INT,

ADD FOREIGN KEY (department\_id) REFERENCES departments(department\_id);

ALTER TABLE courses

ADD COLUMN professor\_ID INT,

ADD FOREIGN KEY (professor\_ID) REFERENCES professors(professor\_ID);

CREATE TABLE enrollments (

enrollment\_id INT PRIMARY KEY,

student\_id INT,

course\_id INT,

grade VARCHAR(2),

FOREIGN KEY (student\_id) REFERENCES students(student\_id),

FOREIGN KEY (course\_id) REFERENCES courses(course\_id)

);

CREATE TABLE schedules (

student\_id INT,

course\_id INT,

start\_time TIME,

end\_time TIME,

room\_number VARCHAR(20),

PRIMARY KEY (student\_id, course\_id),

FOREIGN KEY (student\_id) REFERENCES students(student\_id),

FOREIGN KEY (course\_id) REFERENCES courses(course\_id)

);

CREATE TABLE departments (

department\_id INT PRIMARY KEY,

department\_name VARCHAR(50)

);

CREATE TABLE professors (

professor\_id INT PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

address VARCHAR(100),

phone\_number VARCHAR(20),

email VARCHAR(50)

);

4. Perform task 3 and write all the SQL queries for SELECT, INSERT, UPDATE, and DELETE statements of the system's register and login pages (15 points).

**YOUR ANSWER**:

INSERT INTO students VALUES

(6187231, 'Camron', 'Cisneros', '9393 Elm St', '561-587-5123','camron.cisneros@gmail.com', '2001-08-03'),

(1234567, 'Sierra', 'Justus', '8181 Yahoo Ave', '978-489-9959','sierra.justus@gmail.com', '2001-08-17'),

(8765432, 'Evan', 'Cisneros', '8989 Glee St', '561-587-5124','evan.cisneros@gmail.com', '2005-04-07'),

(0234567, 'Alex', 'Justus', '8987 Yahoo Ave', '978-489-918','sierra.justus@gmail.com', '2001-08-17');

INSERT INTO courses VALUES

  (1, 'Introduction to Computer Science', 'Dr. Smith', '09:00:00', '10:30:00', 'Room 101'),

  (2, 'Data Science Fundamentals', 'Dr. Johnson', '13:00:00', '14:30:00', 'Room 201'),

  (3, 'Database Management Systems', 'Prof. Adams', '11:00:00', '12:30:00', 'Room 301')

INSERT INTO enrollments VALUES

  (1, 6187231, 1, 'A'),

  (2, 8765432, 1, 'B'),

  (3, 1234567, 2, 'A'),

  (4, 0234567, 3, 'A+');

INSERT INTO schedules VALUES

  (6187231, 1, '09:00:00', '10:30:00', 'Room 101'),

  (8765432, 1, '09:00:00', '10:30:00', 'Room 101'),

  (1234567, 2, '13:00:00', '14:30:00', 'Room 201'),

  (0234567, 3, '17:00:00', '19:00:00', 'Room 305');

INSERT INTO departments VALUES

  (1, 'Computer Science'),

  (2, 'Data Science'),

  (3, 'Business Administration');

INSERT INTO professors VALUES

  (1, 'Alex', 'Doe', '123 Elm St.', '555-9876', 'alex.doe@email.com'),

  (2, 'Emily', 'Smith', '456 Oak St.', '555-5432', 'emily.smith@email.com'),

  (3, 'Jake', 'Longley', '789 Tree Ave.', '555-1597', 'jake.longley@aol.com');

UPDATE courses

SET instructor\_name = 'Alex Doe'

WHERE course\_id = 1;

UPDATE courses

SET instructor\_name = 'Emily Smith'

WHERE course\_id = 2;

UPDATE courses

SET instructor\_name = 'Jake Longley',

start\_time = '17:00:00',

end\_time = '19:00:00',

room\_number = 'Room 305'

WHERE course\_id = 3;

UPDATE courses

SET professor\_ID = 1,

department\_ID = 1

WHERE course\_ID = 1;

UPDATE courses

SET professor\_ID = 2,

department\_ID = 2

WHERE course\_ID = 2;

UPDATE courses

SET professor\_ID = 3,

department\_ID = 3

WHERE course\_ID = 3;

UPDATE courses

SET department\_id = 1

WHERE course\_id = 1;

UPDATE courses

SET department\_id = 2

WHERE course\_id = 2;

UPDATE courses

SET department\_id = 3

WHERE course\_id = 3;

UPDATE students

SET last\_name = 'Cisneros'

WHERE student\_id = 1234567;

UPDATE students

SET address = '11750 14th St'

WHERE student\_id = 6187231;

DELETE FROM students

WHERE student\_id = 1234567;

INSERT INTO enrollments (enrollment\_id, student\_id, course\_id, grade)

VALUES (5, 6187231, 2, 'C');

INSERT INTO schedules (student\_id, course\_id, start\_time, end\_time, room\_number)

VALUES (6187231, 2, '13:00:00', '14:30:00', 'Room 201');

UPDATE students

SET pass = 'pass'

WHERE last\_name = 'Cisneros' OR last\_name= 'Justus';

SELECT first\_name AS "First",last\_name AS "Last Name"

FROM students;

SELECT \*

FROM students

WHERE last\_name = 'Cisneros';

SELECT s.student\_id, s.first\_name, s.last\_name, d.department\_name

FROM students s

JOIN enrollments e ON s.student\_ID = e.student\_ID

JOIN courses c ON e.course\_ID = c.course\_ID

JOIN departments d ON c.department\_id = d.department\_id;

SELECT p.first\_name "First", p.last\_name "Last", c.course\_name "Course"

FROM professors p

JOIN courses c ON p.professor\_id = c.professor\_id;

SELECT c.course\_name, s.start\_time, s.end\_time, s.room\_number

FROM schedules s

JOIN courses c ON s.course\_id = c.course\_id

WHERE s.student\_id = 6187231;

SELECT c.course\_name, e.grade, d.department\_name AS Department, CONCAT(p.first\_name, ' ', p.last\_name) AS Teacher

FROM enrollments e

JOIN courses c ON e.course\_id = c.course\_id

JOIN departments d ON c.department\_id = d.department\_id

JOIN professors p ON c.professor\_id = p.professor\_id

WHERE e.student\_id = 6187231;

5. Perform task 4 and write what language did you use? What connectivity API did you use? What is your connection string using the chosen programming language? Why did you use this programming language? Capture the screen of the register and login page of the software application you designed (15 points).

**YOUR ANSWER**:

**I used Python to create the GUI, using PyCharm. I used the psycopg2 to connect the database with the GUI. The connection string I used is as following: def connect\_to\_database():**

**try:**

**conn = psycopg2.connect(**

**host="localhost",**

**port="5415",**

**database="studentregistration",**

**user="postgres",**

**password="J@den.2001"**

**)**

**return conn**

**I used this programming language because I am familiar with this language and with my future in Data Science I will need to know more about how to use Python, so I used it mainly as practice for the future.**

**A screen shot of a phone number

Description automatically generated A screenshot of a computer screen

Description automatically generated**

6. Include all the source codes related to performing SELECT, INSERT, UPDATE, and DELETE Statements for the register and login page. The source should show the SQL commands in MySQL (or another DBMS) and the commands used in your programming language to call these SQL statements (9 points).

**YOUR ANSWER**:

**import tkinter as tk**

**import psycopg2**

**# Function to connect to the PostgreSQL database**

**def connect\_to\_database():**

**try:**

**conn = psycopg2.connect(**

**host="localhost",**

**port="5415",**

**database="studentregistration",**

**user="postgres",**

**password="J@den.2001"**

**)**

**return conn**

**except psycopg2.Error as e:**

**print(f"Error connecting to the database: {e}")**

**return None**

**# Function to handle the login button click**

**def login():**

**student\_id = student\_id\_entry.get()**

**password = password\_entry.get()**

**# Connect to the database**

**conn = connect\_to\_database()**

**if conn is not None:**

**cursor = conn.cursor()**

**# Execute a SELECT query to check if the student ID and password match in the database**

**query = f"SELECT \* FROM students WHERE student\_id = {student\_id} AND pass = '{password}'"**

**cursor.execute(query)**

**result = cursor.fetchone()**

**if result:**

**# TODO: Navigate to the home page**

**login\_register\_frame.pack\_forget()**

**home\_page(student\_id)**

**else:**

**print("Invalid credentials")**

**# Close the cursor and connection**

**cursor.close()**

**conn.close()**

**# Function to handle the register button click**

**def register():**

**# Connect to the database**

**conn = connect\_to\_database()**

**if conn is not None:**

**cursor = conn.cursor()**

**# Hide the login/register frame**

**login\_register\_frame.pack\_forget()**

**# Show the registration frame**

**# Create and configure the registration page**

**registration\_frame = tk.Frame(window)**

**# Add registration page elements (labels, entry fields, buttons, etc.)**

**student\_id\_label\_reg = tk.Label(registration\_frame, text="Student ID:")**

**student\_id\_label\_reg.pack()**

**student\_id\_entry\_reg = tk.Entry(registration\_frame)**

**student\_id\_entry\_reg.pack()**

**first\_name\_label = tk.Label(registration\_frame, text="First Name:")**

**first\_name\_label.pack()**

**first\_name\_entry = tk.Entry(registration\_frame)**

**first\_name\_entry.pack()**

**last\_name\_label = tk.Label(registration\_frame, text="Last Name:")**

**last\_name\_label.pack()**

**last\_name\_entry = tk.Entry(registration\_frame)**

**last\_name\_entry.pack()**

**address\_label = tk.Label(registration\_frame, text="Address:")**

**address\_label.pack()**

**address\_entry = tk.Entry(registration\_frame)**

**address\_entry.pack()**

**phone\_number\_label = tk.Label(registration\_frame, text="Phone Number:")**

**phone\_number\_label.pack()**

**phone\_number\_entry = tk.Entry(registration\_frame)**

**phone\_number\_entry.pack()**

**email\_label = tk.Label(registration\_frame, text="Email:")**

**email\_label.pack()**

**email\_entry = tk.Entry(registration\_frame)**

**email\_entry.pack()**

**dob\_label = tk.Label(registration\_frame, text="Date of Birth:")**

**dob\_label.pack()**

**dob\_entry = tk.Entry(registration\_frame)**

**dob\_entry.pack()**

**password\_label\_reg = tk.Label(registration\_frame, text="Password:")**

**password\_label\_reg.pack()**

**password\_entry\_reg = tk.Entry(registration\_frame, show="\*")**

**password\_entry\_reg.pack()**

**register\_button\_reg = tk.Button(registration\_frame, text="Register",**

**command=lambda: submit\_registration(student\_id\_entry\_reg.get(),**

**first\_name\_entry.get(),**

**last\_name\_entry.get(),**

**address\_entry.get(),**

**phone\_number\_entry.get(),**

**email\_entry.get(),**

**dob\_entry.get(),**

**password\_entry\_reg.get(),**

**registration\_frame))**

**register\_button\_reg.pack()**

**registration\_frame.pack()**

**back\_button = tk.Button(registration\_frame, text="Back", command=lambda: go\_back\_login(registration\_frame))**

**back\_button.pack()**

**# Close the cursor and connection**

**cursor.close()**

**conn.close()**

**def go\_back\_login(previous\_frame):**

**# Hide the registration frame**

**previous\_frame.pack\_forget()**

**# Show the login/register frame**

**login\_register\_frame.pack()**

**# Function to handle the submit registration button click**

**def submit\_registration(student\_id, first\_name, last\_name, address, phone\_number, email, date\_of\_birth, password,registration\_frame):**

**# Get the student information from the entry fields**

**"""**

**student\_id = student\_id\_entry\_reg.get()**

**first\_name = first\_name\_entry.get()**

**last\_name = last\_name\_entry.get()**

**address = address\_entry.get()**

**phone\_number = phone\_number\_entry.get()**

**email = email\_entry.get()**

**date\_of\_birth = dob\_entry.get()**

**password = password\_entry\_reg.get()**

**"""**

**# Connect to the database**

**conn = connect\_to\_database()**

**if conn is not None:**

**cursor = conn.cursor()**

**# Execute an INSERT query to add the new student to the database**

**query = f"INSERT INTO students (student\_id, first\_name, last\_name, address, phone\_number, email, date\_of\_birth, pass) " \**

**f"VALUES ({student\_id}, '{first\_name}', '{last\_name}', '{address}', '{phone\_number}', '{email}', '{date\_of\_birth}', '{password}')"**

**cursor.execute(query)**

**# Commit the changes to the database**

**conn.commit()**

**# Close the cursor and connection**

**cursor.close()**

**conn.close()**

**print("Registration successful")**

**# Hide the registration frame**

**registration\_frame.pack\_forget()**

**# Show the login/register frame**

**login\_register\_frame.pack()**

**# Function to create and configure the home page**

**def home\_page(student\_id):**

**home\_frame = tk.Frame(window)**

**# Query the database for the student's schedule**

**conn = connect\_to\_database()**

**if conn is not None:**

**cursor = conn.cursor()**

**query = f"SELECT c.course\_ID, c.course\_name, c.instructor\_name, c.start\_time, c.end\_time " \**

**f"FROM courses c " \**

**f"JOIN enrollments e ON c.course\_id = e.course\_id " \**

**f"WHERE e.student\_id = {student\_id}"**

**cursor.execute(query)**

**result = cursor.fetchall()**

**cursor.close()**

**conn.close()**

**if result:**

**for course in result:**

**course\_label = tk.Label(home\_frame, text=f"Course ID: {course[0]}\n"**

**f"Course Name: {course[1]}\n"**

**f"Instructor: {course[2]}\n"**

**f"Start Time: {course[3]}\n"**

**f"End Time: {course[4]}\n")**

**course\_label.pack()**

**delete\_button = tk.Button(home\_frame, text="Delete",**

**command=lambda course\_id=course[0]: delete\_course(course\_id, student\_id,home\_frame))**

**delete\_button.pack()**

**else:**

**no\_courses\_label = tk.Label(home\_frame, text="You currently have no courses, please add a course.")**

**no\_courses\_label.pack()**

**# Add Course button**

**add\_course\_button = tk.Button(home\_frame, text="Add Course", command=lambda: add\_course(home\_frame, student\_id))**

**add\_course\_button.pack()**

**# Update Information button**

**update\_info\_button = tk.Button(home\_frame, text="View Personal Information",**

**command=lambda: update\_information(student\_id,home\_frame))**

**update\_info\_button.pack()**

**# Log out button**

**logout\_button = tk.Button(home\_frame, text="Log Out", command=lambda: logout(home\_frame))**

**logout\_button.pack()**

**home\_frame.pack()**

**def delete\_course(course\_id, student\_id,previous\_frame):**

**previous\_frame.pack\_forget()**

**# Connect to the database**

**conn = connect\_to\_database()**

**if conn is not None:**

**cursor = conn.cursor()**

**# Execute a DELETE query to remove the course from the student's schedule**

**query = f"DELETE FROM enrollments WHERE student\_id = {student\_id} AND course\_id = {course\_id}"**

**cursor.execute(query)**

**# Commit the changes to the database**

**conn.commit()**

**# Close the cursor and connection**

**cursor.close()**

**conn.close()**

**print("Course deleted from schedule")**

**# Refresh the home page**

**home\_page(student\_id)**

**# Function to handle the add course button click**

**def add\_course(previous\_frame, student\_id):**

**previous\_frame.pack\_forget()**

**add\_course\_frame = tk.Frame(window)**

**# Query the database for available courses that the student doesn't already have**

**conn = connect\_to\_database()**

**if conn is not None:**

**cursor = conn.cursor()**

**query = f"SELECT c.course\_id, c.course\_name, c.instructor\_name, c.start\_time, c.end\_time " \**

**f"FROM courses c " \**

**f"WHERE c.course\_id NOT IN " \**

**f"(SELECT e.course\_id FROM enrollments e WHERE e.student\_id = {student\_id})"**

**cursor.execute(query)**

**result = cursor.fetchall()**

**cursor.close()**

**conn.close()**

**if result:**

**for course in result:**

**course\_info = f"Course ID: {course[0]}\n" \**

**f"Course Name: {course[1]}\n" \**

**f"Instructor: {course[2]}\n" \**

**f"Start Time: {course[3]}\n" \**

**f"End Time: {course[4]}"**

**course\_label = tk.Label(add\_course\_frame, text=course\_info)**

**course\_label.pack()**

**# Add Course button**

**add\_button = tk.Button(add\_course\_frame, text="Add", command=lambda course\_id=course[0]: add\_course\_to\_schedule(student\_id, course\_id, add\_course\_frame))**

**add\_button.pack()**

**else:**

**no\_courses\_label = tk.Label(add\_course\_frame, text="No available courses to add.")**

**no\_courses\_label.pack()**

**# Back button**

**back\_button = tk.Button(add\_course\_frame, text="Back", command=lambda: go\_back(add\_course\_frame, student\_id))**

**back\_button.pack()**

**add\_course\_frame.pack()**

**# Function to handle the add course to schedule functionality**

**def add\_course\_to\_schedule(student\_id, course\_id, previous\_frame):**

**# Connect to the database**

**conn = connect\_to\_database()**

**if conn is not None:**

**cursor = conn.cursor()**

**# Generate a new enrollment\_id**

**cursor.execute("SELECT MAX(enrollment\_id) FROM enrollments")**

**result = cursor.fetchone()**

**enrollment\_id = result[0] + 1 if result[0] else 1**

**# Execute an INSERT query to add the course to the student's schedule with the generated enrollment\_id**

**query = f"INSERT INTO enrollments (enrollment\_id, student\_id, course\_id) " \**

**f"VALUES ({enrollment\_id}, {student\_id}, {course\_id})"**

**cursor.execute(query)**

**# Commit the changes to the database**

**conn.commit()**

**# Close the cursor and connection**

**cursor.close()**

**conn.close()**

**print("Course added to schedule")**

**# Return to the home page**

**previous\_frame.pack\_forget()**

**home\_page(student\_id)**

**# Function to handle the update information button click**

**def update\_information(student\_id,previous\_frame):**

**previous\_frame.pack\_forget()**

**# Create the update information frame**

**view\_info\_frame = tk.Frame(window)**

**# Query the database to retrieve the student's information**

**conn = connect\_to\_database()**

**if conn is not None:**

**cursor = conn.cursor()**

**query = f"SELECT \* FROM students WHERE student\_id = {student\_id}"**

**cursor.execute(query)**

**result = cursor.fetchone()**

**cursor.close()**

**conn.close()**

**if result:**

**# Display the student's information**

**student\_info = f"Student ID: {result[0]}\n" \**

**f"First Name: {result[1]}\n" \**

**f"Last Name: {result[2]}\n" \**

**f"Address: {result[3]}\n" \**

**f"Phone Number: {result[4]}\n" \**

**f"Email: {result[5]}\n" \**

**f"Date of Birth: {result[6]}"**

**student\_info\_label = tk.Label(view\_info\_frame, text=student\_info)**

**student\_info\_label.pack()**

**# Update Information button**

**update\_button = tk.Button(view\_info\_frame, text="Update", command=lambda: update\_student\_information(student\_id, view\_info\_frame))**

**update\_button.pack()**

**else:**

**no\_info\_label = tk.Label(view\_info\_frame, text="No information found for the student.")**

**no\_info\_label.pack()**

**# Back button**

**back\_button = tk.Button(view\_info\_frame, text="Back", command=lambda: go\_back(view\_info\_frame, student\_id))**

**back\_button.pack()**

**view\_info\_frame.pack()**

**# Function to handle the update student information functionality**

**def update\_student\_information(student\_id, previous\_frame):**

**previous\_frame.pack\_forget()**

**# Create and configure the update frame**

**update\_frame = tk.Frame(window)**

**# Address**

**address\_label = tk.Label(update\_frame, text="Address:")**

**address\_label.pack()**

**address\_entry = tk.Entry(update\_frame)**

**address\_entry.pack()**

**# Phone Number**

**phone\_number\_label = tk.Label(update\_frame, text="Phone Number:")**

**phone\_number\_label.pack()**

**phone\_number\_entry = tk.Entry(update\_frame)**

**phone\_number\_entry.pack()**

**# Email**

**email\_label = tk.Label(update\_frame, text="Email:")**

**email\_label.pack()**

**email\_entry = tk.Entry(update\_frame)**

**email\_entry.pack()**

**# Update Information button**

**update\_button = tk.Button(update\_frame, text="Update", command=lambda: execute\_update(student\_id, address\_entry.get(), phone\_number\_entry.get(), email\_entry.get(), update\_frame))**

**update\_button.pack()**

**# Back button**

**back\_button = tk.Button(update\_frame, text="Back", command=lambda: go\_back(update\_frame, student\_id))**

**back\_button.pack()**

**update\_frame.pack()**

**def execute\_update(student\_id, address, phone\_number, email, previous\_frame):**

**# Connect to the database**

**conn = connect\_to\_database()**

**if conn is not None:**

**cursor = conn.cursor()**

**# Execute an UPDATE query to update the student's information**

**query = f"UPDATE students " \**

**f"SET address = '{address}', " \**

**f"phone\_number = '{phone\_number}', " \**

**f"email = '{email}' " \**

**f"WHERE student\_id = {student\_id}"**

**cursor.execute(query)**

**# Commit the changes to the database**

**conn.commit()**

**# Close the cursor and connection**

**cursor.close()**

**conn.close()**

**print("Information updated")**

**# Return to the home page**

**previous\_frame.pack\_forget()**

**home\_page(student\_id)**

**# Function to handle the log out functionality**

**def logout(previous\_frame):**

**previous\_frame.pack\_forget()**

**login\_register\_frame.pack()**

**# Function to handle going back to the previous frame**

**def go\_back(previous\_frame, student\_id):**

**previous\_frame.pack\_forget()**

**home\_page(student\_id)**

**# Create the main window**

**window = tk.Tk()**

**# Create and configure the login/register page**

**login\_register\_frame = tk.Frame(window)**

**# Add login/register page elements (labels, entry fields, buttons, etc.)**

**student\_id\_label = tk.Label(login\_register\_frame, text="Student ID:")**

**student\_id\_label.pack()**

**student\_id\_entry = tk.Entry(login\_register\_frame)**

**student\_id\_entry.pack()**

**password\_label = tk.Label(login\_register\_frame, text="Password:")**

**password\_label.pack()**

**password\_entry = tk.Entry(login\_register\_frame, show="\*")**

**password\_entry.pack()**

**login\_button = tk.Button(login\_register\_frame, text="Login", command=login)**

**login\_button.pack()**

**register\_button = tk.Button(login\_register\_frame, text="Register", command=register)**

**register\_button.pack()**

**login\_register\_frame.pack()**

**# Run the GUI event loop**

**window.mainloop()**

7. Write any limitations and bugs your system includes. You can add a capture screen of your issues in the following section (6 points).

**YOUR ANSWER**:

**SQL queries injected into user input without proper sanitization exposes vulnerability to attacks. Duplicate student IDs can be registered due to issues with the registration functionality. Assumptions about valid course information retrieval are made on the home page. Insufficient error handling and user feedback hinder issue identification and resolution.**

8. Write any obstacles, issues, problems, difficulties, etc., you encountered during the design, development, implementation, and demonstration of the Student Registrations System at ABC University (6 points).

**YOUR ANSWER**:

**The lack of user session management and authentication tokens, plain text password storage, incomplete database schema, and absence of concurrency and race condition handling were among the issues I encountered. Furthermore, the GUI design lacks appeal and is rudimentary, input validation is absent, and there is no mechanism for logging or auditing.**

9. Make a video screen of your working system in which you demonstrate your database design, diagrams, tables, relationships, and software application for register and login functionalities. First, you should add your name, surname, and other required information to the MySQL database through your register and login GUI of your software application. Next, you must perform INSERT, UPDATE, DELETE, and SELECT statements via the GUI. Once you have made each statement, go to the MySQL database and show that your command has been successfully operated.

Add your project source code, all the documents, excluding this Word file, and all other related files of your project, including your SQL commands, images, etc., in a ZIP file and upload three files to Canvas: 1) a video demonstration file, 2) a ZIP file including all the files of your project, the documents, and images, and 3) the filled in version of this word document to Canvas portal (21 points).

**YOUR ANSWER**:

**Do not send your source code via the email address or Canvas message** (1 point). You will lose all 25 points for the individual project if you send your tasks via email or Canvas message.

End of the document.