Project Report on

Student management using python and SQL

Submitted by

LEVRY NAKA HENRI-MICHEL

25MCI10388

Under the guidance of

Mr.Sachin Raj

in partial fulfilment for the award of the degree of

MASTER OF COMPUTERAPPLICATIONS

ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Chandigarh University

OCT 2025

Certificate

This is to certify that LEVRY NAKA HENRI-MICHEL, a student of Master in Artificial Intelligence & Machin Learning (AI & ML), has successfully completed the Minor Project titled “STUDENT MANAGEMENT” under the esteemed guidance of Mr. sachin raj, Assistant Professor, University Institute of Computing (UIC), Chandigarh University. This project was undertaken as a part of the academic curriculum and is submitted in partial fulfilment of the requirements for the MAM program. The work presented in this project is a result of individual research, diligent effort, and dedication, demonstrating the student’s ability to apply theoretical knowledge to practical problem-solving. The project “STUDENT MANAGEMENT” is a digital platform that stores, organizes, and processes student-related information such as enrollment, attendance, grades, examinations, and performance reports. It helps teachers, administrators, and students access and manage educational data in a systematic way.

Mr.Sachin Raj

Assistant Professor

University Institute of Computing

Chandigarh University

Acknowledgement

I would like to express my sincere gratitude to Chandigarh University and the University Institute of Computing (UIC) for providing me with the opportunity to undertake this project, "AI Reel Generator" I extend my heartfelt appreciation to my esteemed mentor, Mr.Sachin Raj, for her invaluable guidance, continuous support, and insightful feedback throughout the project. His expertise in Python Programming played a crucial role in the successful completion of this project. I am also grateful to my friends and peers for their encouragement and discussions, which helped refine my approach. Lastly, I thank my family for their unwavering support and motivation during this research. This project has been an incredible learning experience, and I hope it serves as a foundation for further exploration in Python Programming.

MCA (Artificial Intelligence & Machine Learning)

Chandigarh University

Table Of Content

Certificate 2

Acknowledgement 3

1. Introduction 5

2. Objectives 6

5.1. RESULT 18

5.2. DATABASE 18

5.3 GRADES 19

5.4 PROFESSOR 19

5.5 STUDENT 20

6. Problems Identified 20

7. Solutions Implemented 20

8. Limitations 20

9. Conclusion 21

10. References 21

1. Introduction

The Student Management System is a Python-based application designed to help educational institutions efficiently manage student data, academic performance, and related information. It replaces traditional manual recordkeeping with an automated, secure, and user-friendly system that allows professors to register, update, and evaluate students with ease. This system ensures data accuracy, minimizes administrative workload, and provides quick access to important student statistics such as grades, CGPA, and ranking.

2. Objectives

The main objectives of the project are:

- To develop an efficient system for managing student records digitally.
- To simplify operations such as adding, editing, deleting, and searching students.
- To calculate students’ CGPA automatically based on their grades.
- To maintain and display student rankings according to academic performance.
- To ensure secure professor authentication before accessing the system.
- To provide statistical insights such as average CGPA, number of students, and performance distribution.

The overarching aim is to minimize manual production effort, providing a scalable and efficient tool for rapid vertical video creation.

3. Tools and Technologies Used

Component	Specification
Operating System	Windows / Linux / macOS
Programming Language	Python 3.8+
Database	MySQL 8.0 or later
Python Libraries	mysql-connector-python, datetime
IDE / Editor	VS Code, PyCharm, or any Python IDE

4. Implementation Steps

Step 1: Start

Step 2: Initialize system and connect to MySQL database Step

3: Display login screen Step

4: Professor enters username and password Step

5: If credentials match → grant access; else retry (max 3 attempts)

Step 6: Display main menu with options

Step 7: Based on user choice:

- Add new student → store details in database
- View students → fetch and display all records
- Search student → query by ID or name
- Edit student → update specific record
- Delete student → remove record after confirmation
- Manage grades → insert/update/delete grade in grade table
- Calculate CGPA → average all grades for each student
- Display ranking → sort students by CGPA in descending order
- Display statistics → compute and show total students, average CGPA, etc.

Step 8: Repeat menu until user chooses Exit

Step 9: Close connection and terminate program

Step 10: End

Explanation:

The algorithm begins by initializing the system and authenticating the professor using a username and password. Once access is granted, the main menu provides various functionalities to manage students. Each menu action interacts with the database to perform the desired CRUD (Create, Read, Update, Delete) operations. For academic evaluation, the program collects grades for each subject and calculates the CGPA using the formula:

CGPA=∑Grades Number of Subjects CGPA = \frac{\sum \text{Grades}}{\text{Number of Subjects}}CGPA=Number of Subjects ∑Grades.

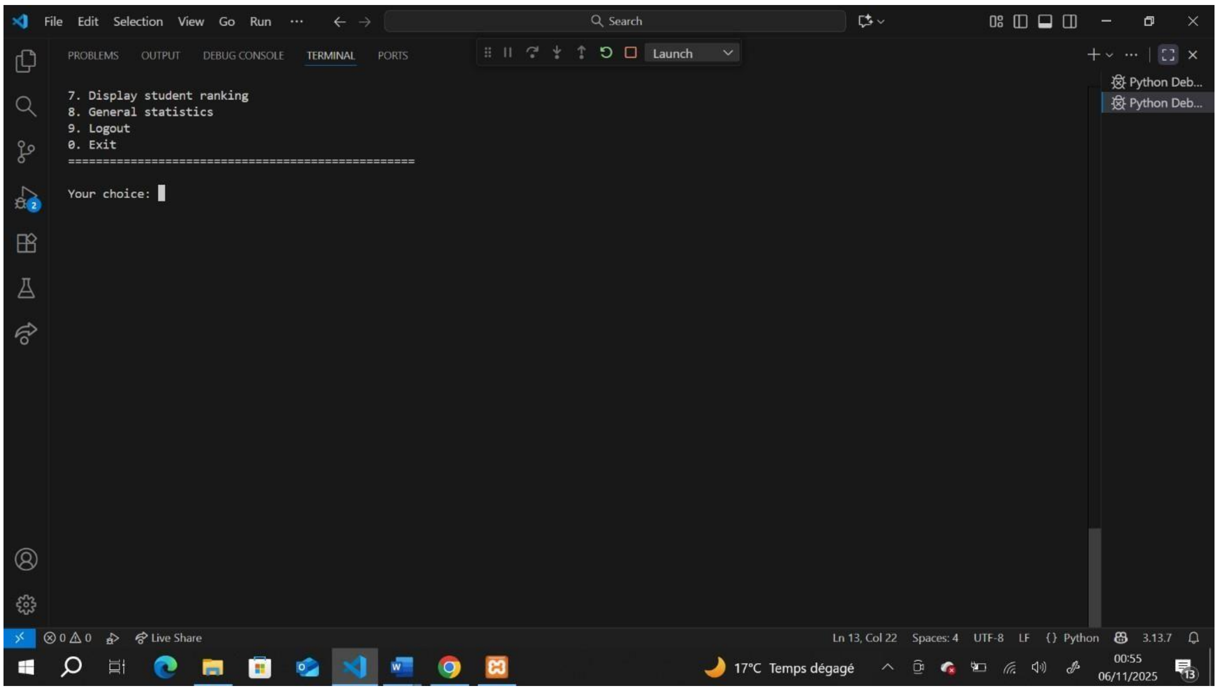
This value is automatically updated in the student’s record. The system can then rank students based on CGPA and display summary statistics such as highest, lowest, and average CGPA. Finally, when the user logs out or exits, all database connections are safely closed, ensuring data integrity and security.

5. Screenshots/Result

```python
import mysql.connector
from datetime import datetime
from db_config import get_connection

class StudentManagementSystem:
    def __init__(self):
        self.current_professor = None

    # ---------- PROFESSOR AUTHENTICATION ----------
    def authenticate(self):
        print("\n" + "="*50)
        print("     PROFESSOR LOGIN")
        print("="*50)

        conn = get_connection()
        cur = conn.cursor(dictionary=True)

        max_attempts = 3
        for attempt in range(max_attempts):
            username = input("\nUsername: ").strip()
            password = input("Password: ").strip()

            cur.execute("SELECT * FROM professors WHERE username=%s AND password=%s", (username, password))
            professor = cur.fetchone()

            if professor:
                self.current_professor = professor
                print(f"\n✓ Login successful! Welcome {professor['name']}")
                conn.close()
                return True
            else:
                remaining = max_attempts - attempt - 1
                if remaining > 0:
                    print(f"\n✗ Invalid credentials. {remaining} attempt(s) left.")
                else:
                    print("\n✗ Maximum attempts reached.")
        conn.close()
        return False

    # ---------- CGPA CALCULATION ----------
    def calculate_cgpa(self, student_id):
        conn = get_connection()
        cur = conn.cursor()
        cur.execute("SELECT AVG(grade) FROM grades WHERE student_id=%s", (student_id,))
        avg = cur.fetchone()[0]
        avg = round(avg, 2) if avg else 0.0
        cur.execute("UPDATE students SET cgpa=%s WHERE student_id=%s", (avg, student_id))
        conn.commit()
        conn.close()
        return avg

    # ---------- MENU ----------
    def display_menu(self):
        print("\n" + "="*50)
        print("      STUDENT MANAGEMENT SYSTEM")
        print("="*50)
        print("\n1. Register new student")
        print("2. View all students")
        print("3. Search student")
        print("4. Edit student")
        print("5. Delete student")
        print("6. Manage student grades")
        print("7. Display student ranking")
        print("8. General statistics")
        print("9. Logout")
        print("0. Exit")
        print("="*50)

    # ---------- ADD STUDENT ----------
    def add_student(self):
        print("\n--- REGISTER NEW STUDENT ---")
        conn = get_connection()
        cur = conn.cursor()

        cur.execute("SELECT MAX(id) FROM students")
        last_id = cur.fetchone()[0]
        new_id = 1 if last_id is None else last_id + 1
        student_id = f"STU{new_id:04d}"

        print(f"\nGenerated ID: {student_id}")
        first_name = input("First Name: ").strip()
        last_name = input("Last Name: ").strip()
        email = input("Email: ").strip()
        phone = input("Phone: ").strip()
        department = input("Department: ").strip()

        cur.execute("""
            INSERT INTO students (student_id, first_name, last_name, email, phone, department, date_registered)
            VALUES (%s, %s, %s, %s, %s, %s, %s)
        """, (student_id, first_name, last_name, email, phone, department, datetime.now()))

        conn.commit()
        conn.close()
        print(f"\n✓ Student {first_name} {last_name} registered successfully! (ID: {student_id})")

          # ---------- VIEW ALL STUDENTS ----------
          def display_all_students(self):
              conn = get_connection()
              cur = conn.cursor(dictionary=True)
              cur.execute("SELECT * FROM students ORDER BY id")
              students = cur.fetchall()
              conn.close()

              if not students:
                  print("\n⚠ No students registered.")
                  return

              print("\n" + "="*100)
              print(f"{'ID':<10} {'First Name':<15} {'Last Name':<15} {'Department':<15} {'CGPA':<8} {'Email':<25}")
              print("="*100)
              for s in students:
                  print(f"{s['student_id']:<10} {s['first_name']:<15} {s['last_name']:<15} {s['department']:<15} {s['cgpa']:<8} {s['email']:<25}")
              print("="*100)
              print(f"\nTotal: {len(students)} student(s)")

          # ---------- SEARCH STUDENT ----------
          def search_student(self):
              print("\n--- SEARCH STUDENT ---")
              print("1. Search by ID")
              print("2. Search by Name")
              choice = input("\nYour choice: ").strip()
              conn = get_connection()
              cur = conn.cursor(dictionary=True)

              if choice == "1":
                  sid = input("Student ID: ").strip().upper()
                  cur.execute("SELECT * FROM students WHERE student_id=%s", (sid,))
                  s = cur.fetchone()
                  if s:
                      self.display_student_details(s)
                  else:
                      print(f"\n✗ No student found with ID: {sid}")

              elif choice == "2":
                  name = input("Name to search: ").strip()
                  cur.execute("SELECT * FROM students WHERE first_name LIKE %s OR last_name LIKE %s", (f"%{name}%", f"%{name}%"))
                  found = cur.fetchall()
                  if found:
                      print(f"\n✓ {len(found)} student(s) found:")
                      for s in found:
                          self.display_student_details(s)
                  else:
                      print(f"\n✗ No student found with name: {name}")
              conn.close()

          # ---------- DISPLAY STUDENT DETAILS ----------
          def display_student_details(self, s):
              print("\n" + "-"*50)
              print(f"ID: {s['student_id']}")
              print(f"Full Name: {s['first_name']} {s['last_name']}")
              print(f"Email: {s['email']}")
              print(f"Phone: {s['phone']}")
              print(f"Department: {s['department']}")
              print(f"CGPA: {s['cgpa']}/10")
              print(f"Registered on: {s['date_registered']}")
              conn = get_connection()
              cur = conn.cursor(dictionary=True)
              cur.execute("SELECT subject, grade FROM grades WHERE student_id=%s", (s['student_id'],))
              grades = cur.fetchall()
              if grades:
                  print("\nGrades:")
                  for g in grades:
                      print(f"  - {g['subject']}: {g['grade']}/10")
              else:
                  print("\nNo grades recorded.")
              conn.close()
              print("-"*50)

          # ---------- EDIT STUDENT ----------
          def edit_student(self):
              conn = get_connection()
              cur = conn.cursor(dictionary=True)
              sid = input("\nStudent ID to edit: ").strip().upper()
              cur.execute("SELECT * FROM students WHERE student_id=%s", (sid,))
              s = cur.fetchone()

              if not s:
                  print("\n✗ No student found.")
                  conn.close()
                  return

              print(f"\nEditing student: {s['first_name']} {s['last_name']}")
              print("Leave blank to keep current value.")

              first_name = input(f"First Name [{s['first_name']}]: ").strip() or s['first_name']
              last_name = input(f"Last Name [{s['last_name']}]: ").strip() or s['last_name']
              email = input(f"Email [{s['email']}]: ").strip() or s['email']
              phone = input(f"Phone [{s['phone']}]: ").strip() or s['phone']
              department = input(f"Department [{s['department']}]: ").strip() or s['department']

              cur.execute("""
                  UPDATE students SET first_name=%s, last_name=%s, email=%s, phone=%s, department=%s WHERE student_id=%s
              """, (first_name, last_name, email, phone, department, sid))
              conn.commit()
              conn.close()
              print("\n✓ Student information updated successfully.")

          # ---------- DELETE STUDENT ----------
          def delete_student(self):
              conn = get_connection()
              cur = conn.cursor(dictionary=True)
              sid = input("\nStudent ID to delete: ").strip().upper()
              cur.execute("SELECT * FROM students WHERE student_id=%s", (sid,))
              s = cur.fetchone()

              if not s:
                  print("\n✗ No student found.")
                  conn.close()
                  return

              confirm = input(f"\n⚠ Are you sure you want to delete {s['first_name']} {s['last_name']}? (yes/no): ").strip().lower()
              if confirm == "yes":
                  cur.execute("DELETE FROM students WHERE student_id=%s", (sid,))
                  conn.commit()
                  print("\n✓ Student deleted successfully.")
              else:
```

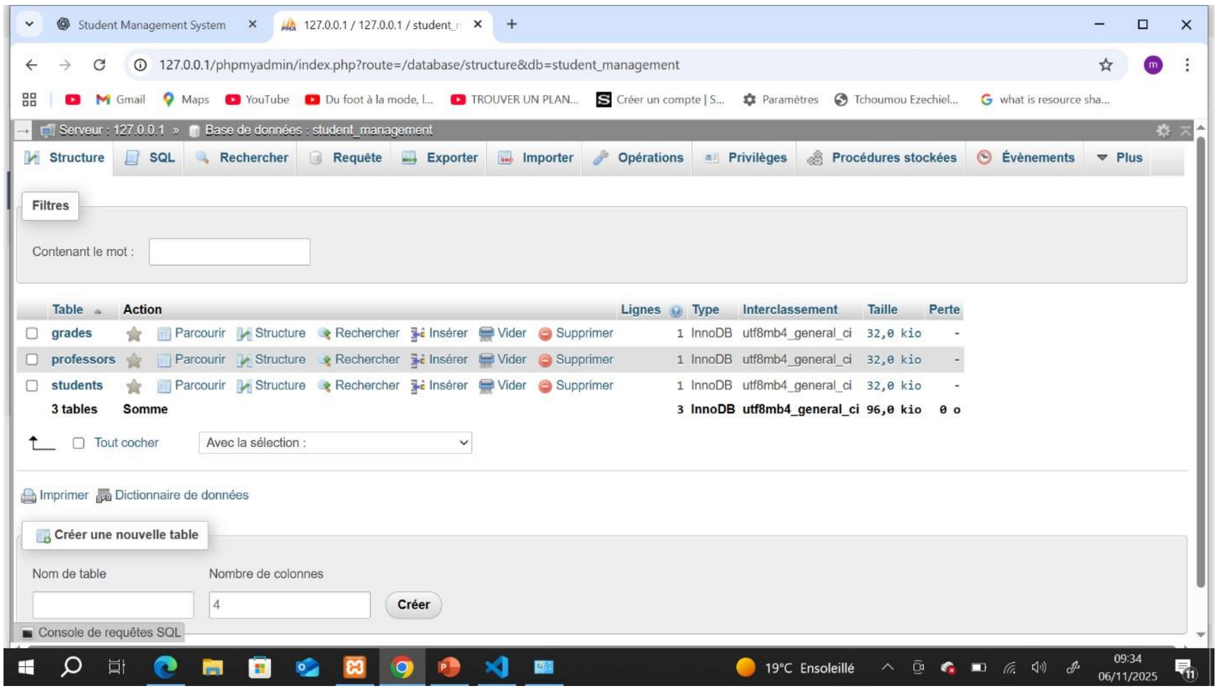
```python
            print("\n✗ Deletion cancelled.")
        conn.close()

    # ---------- MANAGE GRADES ----------
    def manage_grades(self):
        conn = get_connection()
        cur = conn.cursor(dictionary=True)
        sid = input("\nStudent ID: ").strip().upper()
        cur.execute("SELECT * FROM students WHERE student_id=%s", (sid,))
        s = cur.fetchone()

        if not s:
            print("\n✗ No student found.")
            conn.close()
            return

        print(f"\nManaging grades for: {s['first_name']} {s['last_name']}")
        print("\n1. Add/Edit grade")
        print("2. Delete grade")
        print("3. Show all grades")

        choice = input("\nYour choice: ").strip()

        if choice == "1":
            subject = input("Subject: ").strip()
            try:
                grade = float(input("Grade (0-10): ").strip())
                if 0 <= grade <= 10:
                    cur.execute("SELECT * FROM grades WHERE student_id=%s AND subject=%s", (sid, subject))
                    exists = cur.fetchone()
                    if exists:
                        cur.execute("UPDATE grades SET grade=%s WHERE id=%s", (grade, exists['id']))
                    else:
                        cur.execute("INSERT INTO grades (student_id, subject, grade) VALUES (%s, %s, %s)", (sid, subject, grade))
                    conn.commit()
                    avg = self.calculate_cgpa(sid)
                    print(f"\n✓ Grade saved! Updated CGPA: {avg}/10")
                else:
                    print("\n✗ Grade must be between 0 and 10.")
            except ValueError:
                print("\n✗ Invalid input.")

        elif choice == "2":
            subject = input("Subject to delete: ").strip()
            cur.execute("DELETE FROM grades WHERE student_id=%s AND subject=%s", (sid, subject))
            conn.commit()
            avg = self.calculate_cgpa(sid)
            print(f"\n✓ Grade deleted. Updated CGPA: {avg}/10")

        elif choice == "3":
            cur.execute("SELECT * FROM grades WHERE student_id=%s", (sid,))
            grades = cur.fetchall()
            if grades:
                print("\nGrades:")
                for g in grades:
                    print(f"  - {g['subject']}: {g['grade']}/10")
                print(f"\nCGPA: {s['cgpa']}/10")
            else:
                print("\n⚠ No grades recorded.")
        conn.close()

    # ---------- RANKING ----------
    def display_ranking(self):
        conn = get_connection()
        cur = conn.cursor(dictionary=True)
        cur.execute("SELECT * FROM students ORDER BY cgpa DESC")
        students = cur.fetchall()
        conn.close()

        if not students:
            print("\n⚠ No students registered.")
            return

        print("\n" + "="*80)
        print("     STUDENT RANKING (BY CGPA)")
        print("="*80)
        print(f"{'Rank':<6} {'ID':<10} {'Name':<25} {'Department':<20} {'CGPA':<8}")
        print("="*80)

        for rank, s in enumerate(students, 1):
            fullname = f"{s['first_name']} {s['last_name']}"
            print(f"{rank:<6} {s['student_id']:<10} {fullname:<25} {s['department']:<20} {s['cgpa']:<8}")
        print("="*80)

    # ---------- STATISTICS ----------
    def display_statistics(self):
        conn = get_connection()
        cur = conn.cursor(dictionary=True)
        cur.execute("SELECT COUNT(*) as total FROM students")
        total = cur.fetchone()['total']
        cur.execute("SELECT COUNT(DISTINCT student_id) as with_grades FROM grades")
        with_grades = cur.fetchone()['with_grades']
        cur.execute("SELECT AVG(cgpa) as avg_cgpa, MAX(cgpa) as max_cgpa, MIN(cgpa) as min_cgpa FROM students")
        stats = cur.fetchone()

        print("\n" + "="*50)
        print("      GENERAL STATISTICS")
        print("="*50)
        print(f"Total students: {total}")
        print(f"Students with grades: {with_grades}")
        if stats['avg_cgpa'] is not None:
            print(f"\nAverage CGPA: {stats['avg_cgpa']:.2f}/10")
            print(f"Max CGPA: {stats['max_cgpa']:.2f}/10")
            print(f"Min CGPA: {stats['min_cgpa']:.2f}/10")
        else:
            print("\n⚠ No grades recorded yet.")
        print("="*50)
        conn.close()

    # ---------- MAIN LOOP ----------
    def run(self):
        print("\n" + "*"*50)
        print("    WELCOME TO STUDENT MANAGEMENT SYSTEM")
        print("*"*50)

        if not self.authenticate():
            print("\n✗ Authentication failed. Exiting system.")
            return

        while True:
            self.display_menu()
            choice = input("\nYour choice: ").strip()

            if choice == "1":
                self.add_student()
            elif choice == "2":
                self.display_all_students()
            elif choice == "3":
                self.search_student()
            elif choice == "4":
                self.edit_student()
            elif choice == "5":
                self.delete_student()
            elif choice == "6":
                self.manage_grades()
            elif choice == "7":
                self.display_ranking()
            elif choice == "8":
                self.display_statistics()
            elif choice == "9":
                print(f"\n Logging out, {self.current_professor['username']}!")
                if not self.authenticate():
                    break
            elif choice == "0":
                print("\n Thank you for using the system. Goodbye!")
                break
            else:
                print("\n✗ Invalid choice. Try again.")

            input("\nPress Enter to continue...")

if __name__ == "__main__":
    system = StudentManagementSystem()
    system.run()
```

5.1. RESULT



```
7. Display student ranking
8. General statistics
9. Logout
0. Exit
==================================================

Your choice: 
```

5.2. DATABASE

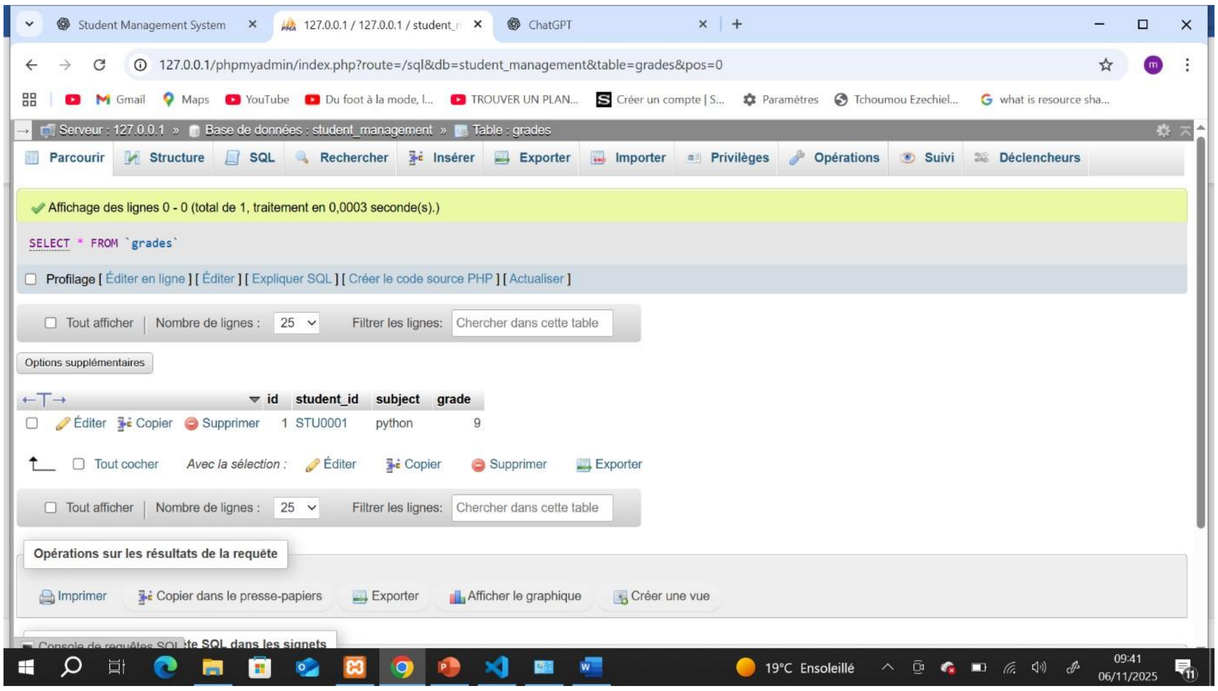


127.0.0.1/phpmyadmin/index.php?route=/database/structure&db=student_management

Serveur : 127.0.0.1 » Base de données : student_management

Table	Action	Lignes	Type	Interclassement	Taille	Perte
grades	Parcourir Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8mb4_general_ci	32,0 kio	-
professors	Parcourir Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8mb4_general_ci	32,0 kio	-
students	Parcourir Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8mb4_general_ci	32,0 kio	-
3 tables	Somme	3	InnoDB	utf8mb4_general_ci	96,0 kio	0 o

5.3 GRADES

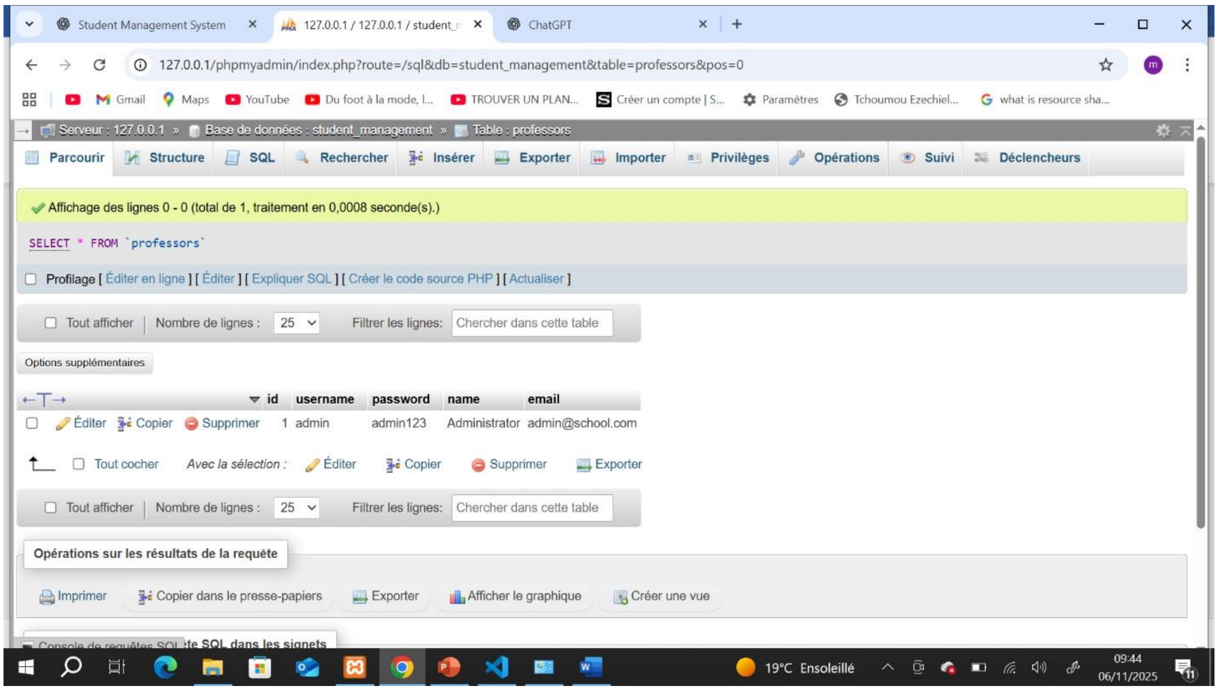


Affichage des lignes 0 - 0 (total de 1, traitement en 0,0003 seconde(s).)

```
SELECT * FROM `grades`
```

id	student_id	subject	grade
1	STU0001	python	9

5.4 PROFESSOR

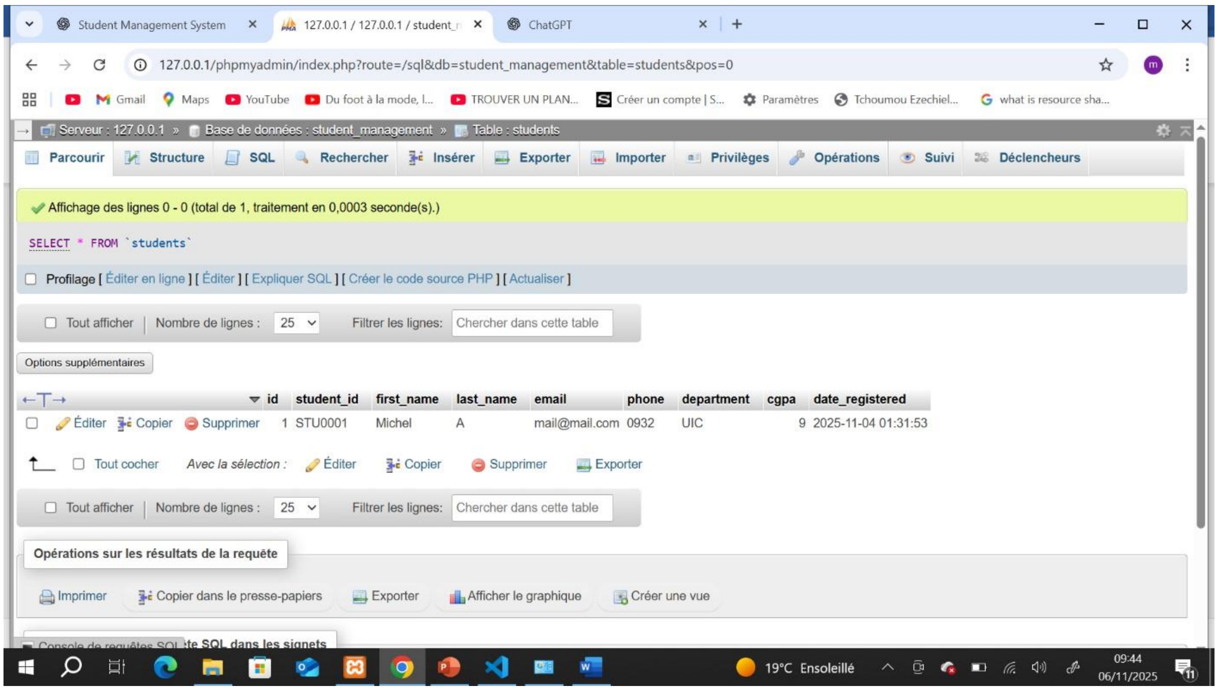


Affichage des lignes 0 - 0 (total de 1, traitement en 0,0008 seconde(s).)

```
SELECT * FROM `professors`
```

id	username	password	name	email
1	admin	admin123	Administrator	admin@school.com

5.5 STUDENT


Student Management System
127.0.0.1 / 127.0.0.1 / student_
ChatGPT
Serveur : 127.0.0.1 » Base de données : student_management » Table : students
Parcourir
Structure
SQL
Rechercher
Insérer
Exporter
Importer
Privilèges
Opérations
Suivi
Déclencheurs
Options supplémentaires
Opérations sur les résultats de la requête
Imprimer
Copier dans le presse-papiers
Exporter
Afficher le graphique
Créer une vue
19°C Ensoleillé
09:44
06/11/2025

127.0.0.1/phpmyadmin/index.php?route=/sql&db=student_management&table=students&pos=0

✓ Affichage des lignes 0 - 0 (total de 1, traitement en 0,0003 seconde(s).)

```
SELECT * FROM `students`
```

Profilage [Éditer en ligne] [Éditer] [Expliquer SQL] [Créer le code source PHP] [Actualiser]

Tout afficher | Nombre de lignes : 25 Filtrer les lignes: Chercher dans cette table

	id	student_id	first_name	last_name	email	phone	department	cgpa	date_registered
Éditer Copier Supprimer	1	STU0001	Michel	A	mail@mail.com	0932	UIC	9	2025-11-04 01:31:53

Tout cocher Avec la sélection : Éditer Copier Supprimer Exporter

6. Problems Identified

1. Manual student record management is time-consuming and prone to human errors.
2. Tracking academic progress manually lacks efficiency and accuracy.
3. Difficulties in maintaining performance statistics and generating rankings.
4. Security issues arise when sensitive academic data is stored in unsecured files.

7. Solutions Implemented

1. Automated all record operations through Python and MySQL integration.
2. Added CGPA calculation and grade management modules.
3. Implemented authentication for professors to prevent unauthorized access.
4. Created ranking and statistical dashboards for data analysis.
5. Designed a user-friendly command-line interface for simplicity and efficiency.

8. Limitations

1. The interface is console-based — no graphical (GUI) version yet.
2. Currently supports single-professor usage at a time.
3. Passwords are stored in plain text (should use hashing for security).
4. Lacks network access for multi-user or remote management.
5. Limited error-handling and input validation.

9. Conclusion

The **Student Management System using Python and MySQL** provides a structured and efficient solution to manage student academic data. It simplifies daily operations for professors, minimizes data entry errors, and automates CGPA computation and performance tracking.

Despite its limitations, the system demonstrates how Python and SQL can be combined to develop reliable database-driven applications for academic institutions.

10. References

1. *Python Official Documentation* – https://docs.python.org/3/
2. *MySQL Documentation* – https://dev.mysql.com/doc/
3. Stack Overflow – Community discussions on Python–MySQL integration.
4. TutorialsPoint – “Python Database Access with MySQL.” 5.
GeeksforGeeks – Python and MySQL CRUD Operations Guide.