# Coursework Submission Declaration

Please read the information below, fill in your details and then submit your completed form on Canvas using the instructions given by your **OnCampus** centre.

Purpose

As stated in the **OnCampus** Programme Handbook, and in your coursework assignment briefs, it is essential that all submitted work is your own. If you submit work that has been completed by somebody else, such as a friend/classmate, Artificial Intelligence (AI) or paid writing service, this is an academic offence and you will not receive a grade for your work. Similarly, copying from another source and claiming it as your own is called plagiarism, which is also an academic offence. **OnCampus** has a responsibility to ensure you have the required language level, skills, and subject knowledge to perform strongly on your degree programme and in your professional career. It is therefore vital that grades are awarded fairly and are based on genuine academic performance.

Please complete the information in the box below to confirm you agree that:

1. At **OnCampus**, we recognise that English may not be your first language. Our assessments are designed to allow you to achieve while continuing to develop your academic English skills. We expect all submitted work to be your own words (apart from in-text quotations), written in a style that reflects your English language level.

2. At any time, you may be asked to attend an interview (known as a ‘viva’) with at least two members of academic staff to discuss the content of your submitted coursework. Vivas may be requested as part of a random check, or if there is suspicion that an academic offence has been committed. **We expect all students to be able to discuss and/or explain the main ideas and vocabulary used in submitted coursework.**

|  |  |
| --- | --- |
| **Student Full Name:** | Aliyan Ali Ramday |
| **Student ID Number:**  (e.g. 12345) | 26924 |
| **Date of Birth:**  (DD/MM/YYYY) | 26/02/2005 |
| **Programme:**  (e.g. UFP) | IYO BSc (Hons) Computer Science |
| **Programme Start Date:** (e.g. September 2024) | 23rd September 2024 |

By submitting this document, you confirm that that you understand and accept all of the information above. This agreement covers all assessed work on your OnCampus programme.

Please tick the box if you are happy for any piece of your completed work to be used for teaching and learning purposes e.g. shared as an example of a completed assignment.

Please note OnCampus reserve the right to use any piece of work for standardisation activities and/or to share with External Examiners.

[Coursework Submission Declation i](#_Toc192455814)

[1. Introduction 1](#_Toc192455815)

[1.1 Project Overview 1](#_Toc192455816)

[1.2 Purpose of the Project 1](#_Toc192455817)

[1.3 Software Engineering Framework 1](#_Toc192455818)

[1.4 Tools and Technologies 1](#_Toc192455819)

[1.5 Scope of the Project 1](#_Toc192455820)

[2. Requirement Analysis 2](#_Toc192455821)

[2.1 Introduction 2](#_Toc192455822)

[2.2 Functional Requirements 2](#_Toc192455823)

[2.3 Non-Functional Requirements 2](#_Toc192455824)

[2.4 User Requirements 3](#_Toc192455825)

[2.5 System Requirements and Cost-Benefit Analysis (Estimated Market Trends) 3](#_Toc192455826)

[2.5.1 Budgeting Table 3](#_Toc192455827)

[2.6 Constraints and Assumptions 4](#_Toc192455828)

[2.7 GANTT Table and Chart 5](#_Toc192455829)

[2.8 Activity Timeline 6](#_Toc192455830)

[2.9 PERT Chart 7](#_Toc192455831)

[3. Implementation of Python GUI with Database Interaction 8](#_Toc192455832)

[3.1 Database Development Using the Database Life Cycle (DBLC) 9](#_Toc192455833)

[3.1.1 Database Initial Study (Requirements Analysis) 9](#_Toc192455834)

[3.1.2 External Schema (Data Description and DDL) 9](#_Toc192455835)

[3.1.3 Conceptual Schema Design 10](#_Toc192455836)

[3.1.4 The Physical Design 15](#_Toc192455837)

[3.2 The flowchart for Program Design 19](#_Toc192455838)

[4. Testing and Implementation 20](#_Toc192455839)

[5. Critical Discussion & Recommendations 27](#_Toc192455840)

# 1. Introduction

## 1.1 Project Overview

This document describes creating a performance data management system that delivers efficient student analytics. The application monitors three academic terms containing three different subjects for each student. The application runs on Python with MySQL as its database solution, which handles data securely and effectively. Through its Python GUI, the application simplifies how users interact with data to make student performance easier to view and access. The core mission of this project involves building a dependable, educational tracking system for teachers that displays real-time achievement data for individual students and whole classes.

1.2 Purpose of the Project

The main goal of this project is to create a system capable of simplifying the viewing and assessment of student subject grades spanning across different educational terms for teachers. The software platform delivers complete student performance reports that support the evaluation of personal achievement and entire classroom achievement levels. The system leverages bar charts and line plots as data visualization tools to deliver straightforward insights about student achievements, enabling performance tracking throughout different periods.

## 1.3 Software Engineering Framework

This software development operates under engineering principles that manifest through systematic approaches to handling intricate problems. Extracting benefits from computational thinking divides the problem into sections that are simpler to handle. This project implements SDLC methodology to carry out requirements gathering and system deployment phases through a logical and practical approach. Technology development follows an SDLC methodology that uses analysis and design stages to implement systems while ensuring testing success and financial and time requirements for reliable user-oriented solutions.

## 1.4 Tools and Technologies

The application uses Python as its development platform because it allows flexible control of database communications while providing broad GUI-building capabilities. The student data storage and management runs on the MySQL backend system, which offers swift data retrieval capabilities. Through the Python framework using Tkinter, students obtain access to interactive platforms that show performance data alongside report generation capabilities. Through the integration of these technologies, a system with efficient project requirements can be developed.

## 1.5 Scope of the Project

The development of this project involves building an entire student performance management system from start to end, including the system's database and interface components. This database provides capabilities to process student information alongside grades and academic performance statistics per academic period. Users will access and examine the necessary data through the interface that provides a platform for interactive functionality to create visual reports. The system maintains scalability and adaptability in its development because it allows instant educational support alongside potential future growth features.

# 2. Requirement Analysis

## 2.1 Introduction

An organized methodology is needed to develop student performance management applications that execute their desired functions at maximum operational efficiency. A thorough analysis of the system functions with non-functional requirements appears in this section to describe essential components, data flow, and operational limitations. These aspects require a clear definition to form a solid base for the following phases involving design, implementation, and testing, leading to functional products with scalability and user-friendly features.

2.2 Functional Requirements

The main operational aspects of this application involve collecting, organizing, and processing academic student performance data from different academic terms. Students need to submit their enrollment data through the system, and all records should be updated with the option to generate performance reports for students and their entire class. Users should have access rights to create new student registries, while an operator can edit current information and delete data that no longer requires them. The system must provide functions for students' grade filtering by academic terms and an interface that produces bar charts and line plots for performance trend investigations. User-friendly and interactive elements must be part of the application because they create an intuitive system that educators and staff members can easily use. The application requires database connectivity as a core functional need to safeguard and retrieve student information through the structured MySQL database system. The system needs basic authentication that allows only authorized personnel with student record management duties to access the system. The system should contain an export functionality that lets users acquire performance reports in structured data formats suitable for documentation and external analysis.

2.3 Non-Functional Requirements

The application needs to follow various non-functional requirements and its core functions to maintain efficiency, reliability, and security standards. An expandable framework should exist within the system, enabling upcoming student and subject enrollment growth alongside academic period increases without significant restructuring. The application must ensure performance efficiency, which allows it to process and retrieve large datasets without creating noticeable delays. The system must ensure superior data integrity through accurate student record storage, which minimizes intentional and unintentional record modifications and deletions. Reliability demands an application that runs without constantly needing repairs or system failure occurrences. The system must support Windows, macOS, and Linux operating systems to give users flexibility when deploying it across different platforms. Security demands access control systems that stop unapproved users from making changes to student records. The system needs an authentication system that employs password protection to authorize modifications from official personnel. The system needs built-in data backup and recovery features that protect against system failures, accidental deletions, and hardware malfunctions.

## 2.4 User Requirements

Educational and support personnel representing the primary users require tools to measure student advancement and create performance reports. The system needs a user-friendly interface without extensive training to let users work effortlessly with student data. The application design allows users to easily transition between different sections to reach needed information without delay. The application needs automated analysis tools for average score calculations per subject and term, enabling educators to detect academic patterns in student achievement. The system needs to allow users to create bespoke reports that will let them analyze student-level results and class averages or results based on particular subjects. Users should have access to filtering systems and sorting features to examine performance metrics through their defined parameters.

## 2.5 System Requirements and Cost-Benefit Analysis (Estimated Market Trends)

The hardware and software resources must be considered for the system to work correctly. The app must be deployed on a machine with at least 4GB RAM, a dual-core processor, and sufficient storage for DB. A stable MySQL server will be utilized for managing student records, and the application’s backend and GUI components will be created using Python.

For the software side, the application will be implemented in Python 3. x and phpMyAdmin (through XAMPP) for database management, with the IDEs being IDLE for Python and VS Code. Tkinter will also be used for UI development, and Matplotlib will be used for data visualization. Using SQL queries to perform operations on Student records in MySQL. We will also add dependencies like Pandas and NumPy for better data processing.

### 2.5.1 Budgeting Table

Although many of the required software tools are open-source and free, a budget has been prepared to account for potential licensing costs, optional premium tools, and necessary hardware.

|  |  |  |
| --- | --- | --- |
| Component | Details | Cost (£) |
| Hardware | Minimum 4GB RAM, Dual-Core Processor, Storage (Assumes personal laptop) | £ - |
| Operating System | Windows / macOS / Linux (Open source) | £ - |
| Development Environment | Python IDLE (Included with Python) | £ - |
| Code Editor | VS Code (Free) | £ - |
| Database Management | MySQL (Free) via XAMPP + phpMyAdmin | £ - |
| GUI Development | Tkinter (Built-in Python Library) | £ - |
| Data Processing | Pandas, NumPy (Free Python Libraries) | £ - |
| Data Visualization | Matplotlib (Free Python Library) | £ - |
| XAMPP | Apache, MySQL, PHP (Free) | £ - |
| Internet & Hosting (Optional) | Domain & Hosting (If online deployment) | £ 50.00 |
| Miscellaneous Costs | Backup storage, optional software tools | £ 20.00 |
| Total Estimated Cost | (Assuming all free tools except hosting) | £70 (Maximum, if online hosting is needed) |

The budget primarily accounts for optional costs, such as web hosting if the system is expanded to an online version. However, for local use, the development cost remains zero, assuming the use of a personal laptop and open-source software.

This structured breakdown ensures that the project remains cost-effective while meeting all functional and non-functional requirements.

## 2.6 Constraints and Assumptions

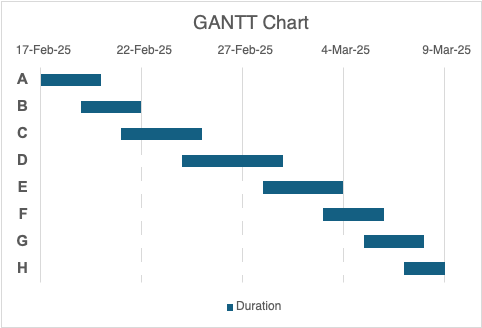
Several factors will constrain the system's development and deployment. The app is mainly targeted towards small to medium educational institutions and probably needs further optimizations to run for institutions with thousands of students. Furthermore, online access in real-time is not part of the first version, as the system will run on standalone machines (local).

Users are expected to be familiar with using a computer and managing student records. The system also assumes that the institutions using it already possess MySQL and that it is configured to work with the application's backend processes.

Fulfilling these needs and constraints helps to guide the application's structural roadmap and ensure the product is functional, reliable, scalable, and user-friendly for the targeted market.

2.7 GANTT Table and Chart

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task ID | Task Name | Start Date | End Date | Duration |
| A | Understanding Application Requirements | 17-Feb-25 | 19-Feb-25 | 3 |
| B | Researching Software Development Framework | 19-Feb-25 | 21-Feb-25 | 3 |
| C | Designing Database Schema | 21-Feb-25 | 24-Feb-25 | 4 |
| D | Developing Python GUI | 24-Feb-25 | 28-Feb-25 | 5 |
| E | Implementing Database Connectivity | 28-Feb-25 | 3-Mar-25 | 4 |
| F | Implementing Student Performance Analysis | 3-Mar-25 | 5-Mar-25 | 3 |
| G | Testing and Debugging | 5-Mar-25 | 7-Mar-25 | 3 |
| H | Completing Application Documentation | 7-Mar-25 | 9-Mar-25 | 3 |

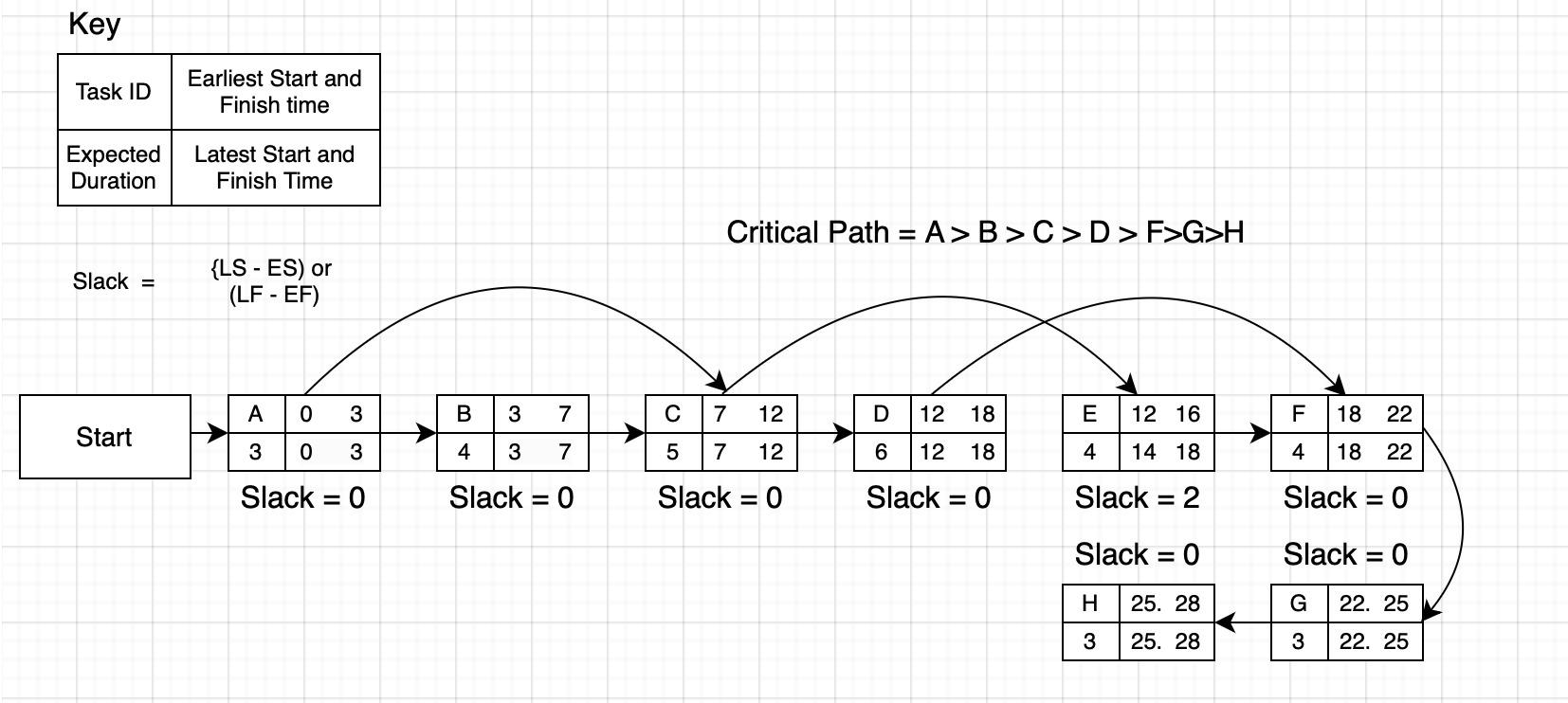


2.8 Activity Timeline

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Task ID | Task Name | Task Predecessor | Optimistic Time (days) | Most Likely Time (days) | Pessimistic Time (days) | Expected Duration (days) | Standard Deviation (days) |
| A | Understanding Application Requirements | - | 2 | 3 | 3 | 3 | 0.17 |
| B | Researching Software Development Framework | A | 3 | 3 | 5 | 4 | 0.33 |
| C | Designing Database Schema | A, B | 3 | 4 | 6 | 5 | 0.50 |
| D | Developing Python GUI | C | 4 | 5 | 7 | 6 | 0.50 |
| E | Implementing Database Connectivity | C | 3 | 4 | 5 | 4 | 0.33 |
| F | Implementing Student Performance Analysis | D, E | 4 | 3 | 6 | 4 | 0.33 |
| G | Testing and Debugging | F | 2 | 3 | 4 | 3 | 0.33 |
| H | Completing Application Documentation | G | 2 | 3 | 3 | 3 | 0.17 |

Activity Timeline for Student Performance Management System ● The activity timeline for the Student Performance Management System has been constructed by arranging the tasks sequentially, showing the dependencies and estimated completion times using the PERT methodology. Understand Application Requirements ( 3 days ):- The project starts with understanding the application requirements, followed by researching the software development framework (4 days). Completing Designing the Database Schema (5 days) enables two parallel tracks: Developing the Python GUI (6 days) and Implementing Database Connectivity (4 days). This is followed by Implementing Student Performance Analysis (4 days), Testing and Debugging (3 days), and Completing Application Documentation (3 days). The dependencies and how they are executed are well thought out; the expected length of each task is used to set expectations for the project duration, and its standard deviation is used to account for any uncertainty.

2.9 PERT Chart



The Program Evaluation and Review Technique (PERT) has been utilized to estimate the completion probability of the project within the designated timeline. PERT accounts for uncertainties in task durations by incorporating three-time estimates for each activity: the optimistic time (O), the most likely time (M), and the pessimistic time (P). The expected duration for each task is calculated using the formula:

*TE=O+4M+P*

6

The project timeline has been assessed based on the critical path, which consists of the longest sequence of dependent tasks: A > B > C > D > F > G > H. The total expected project completion time along this path has been calculated as 28 days.

To evaluate the probability of meeting the deadline, the standard deviation for each task is determined using:

*σ=P−O*

6

By summing up the variances of tasks along the critical path, the total standard deviation of the project timeline is 1.12 days. Using the PERT methodology, the probability of completing the project within the given 28-day deadline is determined through the Z-score formula:

*Z=(Deadline−TotalExpectedTime)*

*∑σ2*

Substituting the values:

*Z=(28−28) =0.00*

*1.12*

A Z-score value of 0.00 gives a 50% probability of completing the project within the given time. The project has a 50% chance of being completed on time, indicating a moderate risk of missing its deadline. While there is a 50% chance of finishing all milestones on due dates, an unforeseen issue could significantly impact scheduling. More uncertain tasks such as "Designing Database Schema" and "Developing Python GUI" add the most variance to the timeline.

It is advised to monitor the execution of these tasks closely to ensure they are done well before the deadline. This will prevent the process of ensuring everything is complete from taking even longer. Implementing proactive resource allocation, focusing on parallel task execution where feasible, and applying early identification of potential bottlenecks can mitigate risks and improve the likelihood of successful project completion.

# 3. Implementation of Python GUI with Database Interaction

This application has been designed to effectively manage data between users and the database: to enter, modify, retrieve, and analyze student performance records through the Student Performance Management System. Built on Python programming, the application features a GUI interface that can connect to a MySQL database through phpMyAdmin. This system enables easy handling of details of students, subjects, and grades for many terms.

To structure the software development methodology, the system is developed in line with the component design specifications, which must satisfy both the functional and technical requirements of the project brief. It consists of four steps: Database Development (via Database Life Cycle (DBLC)) → GUI Integration → Data Visualisation → System Validation. This development methodology assures that the application is built to be scalable, user-friendly, and optimized for students' performance data.

## 3.1 Database Development Using the Database Life Cycle (DBLC)

### 3.1.1 Database Initial Study (Requirements Analysis)

This system will provide a good database to manage students, subjects, and their grades in different terms. It will also keep the data accurate and very easy to access. The system also has the functionality to display three academic terms in a year and three subjects in an academic term and system accessors where the login, student adding, updating records, and performance analysis are implemented.

It includes secure login authentication, data management (CRUD operations) functions, and performance analysis measures. Moreover, the database is built around security, performance, and scalability, a feature that ensures future growth will not compromise performance or require excessive maintenance.

### 3.1.2 External Schema (Data Description and DDL)

The external schema information in the Student Performance Management System describes how data is organized, managed and retrieved. Some of the basic entities this database backs consist of system accessors, pupils, topics and also grades which make sure that student performance values are archived properly and can be retrieved simply.

The SystemAccessors table is responsible for all of the authorized personnel who manage student records and keep the database from going into disarray. The accessors set unique email and password for secure access to the system.

The Students table consists of necessary personal details, such as first and middle names of each student, a unique ID number, and an optional file path to a photo for identification purposes. Student ID starts with 101 for clarity and consistency.

The Subjects table contains all the academic courses offered in the institution. Every subject is recognized with a unique subject code and its corresponding subject name for monitoring students and their grades who participated in it.

The StudentGrades table holds information on student performance, with grades in percentage (0 - 100%) across three academic terms for each subject. Structured as a table, it relates students to their subjects and tracks their academic performance in various terms.

It uses a relational model for storage and retrieval, with logical connections between students, subjects, and grades, and the indexing is done to ensure efficient access to the stored data. In addition, this structure minimizes duplication of data, improves query performance, and keeps student records accurate and up to date.

***System Accessors***

This table stores the credentials of system accessors who have permission to access and modify the database. It ensures secure access to the system.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| AccessorID | INT (AUTO\_INCREMENT, PRIMARY KEY) | Unique identifier for each system accessor. |
| Email | VARCHAR(255) | Unique email address used for login authentication. |
| Password | VARCHAR(255) | Secure password for system access. |

***Students***

This table contains personal and enrollment details of students in the institution. Each student is uniquely identified using a **StudentID**.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| StudentID | INT (AUTO\_INCREMENT, PRIMARY KEY) | Unique identifier for each student. |
| FirstName | VARCHAR(255) | First name of the student. |
| MiddleName | VARCHAR(255) | Middle name of the student (optional). |
| Photo | VARCHAR(255) | File path to the student’s passport-size photo. |

***Subjects***

This table contains information about the academic subjects offered in the institution.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| SubjectCode | VARCHAR(10) (PRIMARY KEY) | Unique code assigned to each subject. |
| SubjectName | VARCHAR(255) | Full name of the subject. |

***StudentGrades***

This table maintains student performance records by storing grades achieved in specific subjects during academic terms.

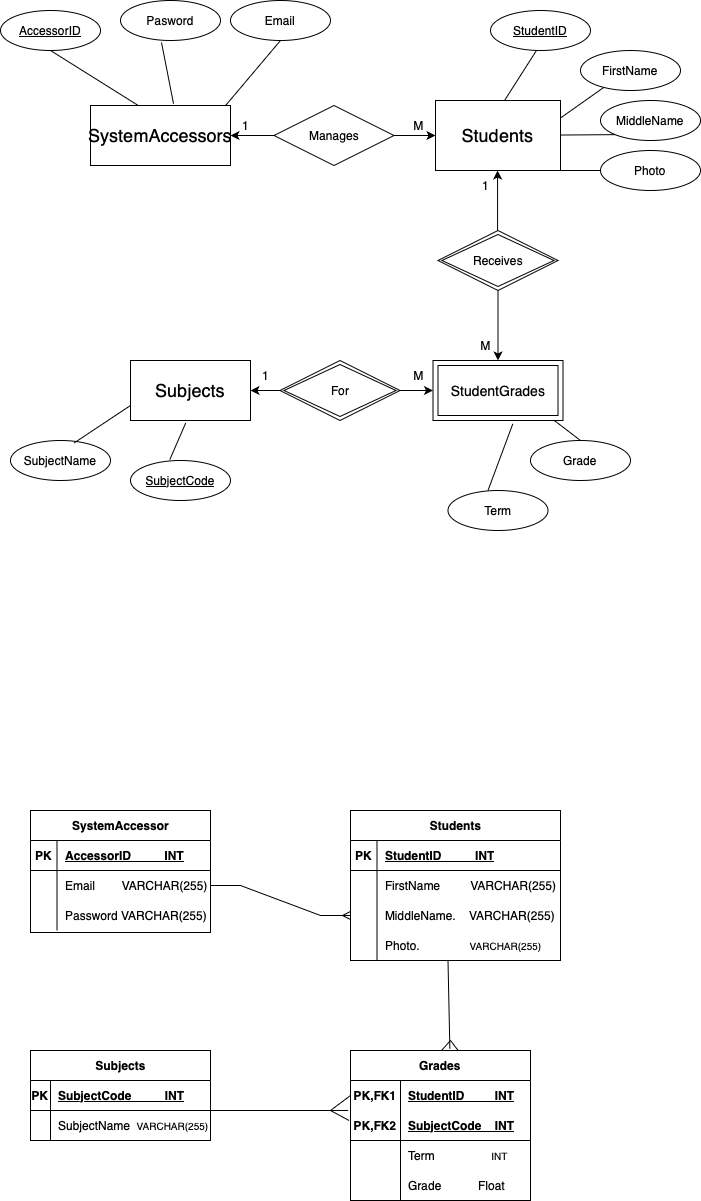
|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| StudentID | INT | Identifies the student associated with the grade. |
| SubjectCode | VARCHAR(10) | The subject for which the grade is recorded. |
| Term | INT (CHECK term BETWEEN 1 AND 3) | The academic term (1, 2, or 3) for which the grade is recorded. |
| Grade | FLOAT (CHECK Grade BETWEEN 0 AND 100) | The student's grade in percentage format (0-100). |

### 3.1.3 Conceptual Schema Design

The conceptual schema represents the logical structure of your database and the interaction of entities to allow the storage and supply of data in a time-efficient manner. The third stage, which is designed to sort out your database, is known as the normalization process. It includes the entity-relationship Diagram (ERD), Relational Model, and Normalization Process. These components provide the foundation to structure student performance data scalable and efficiently.

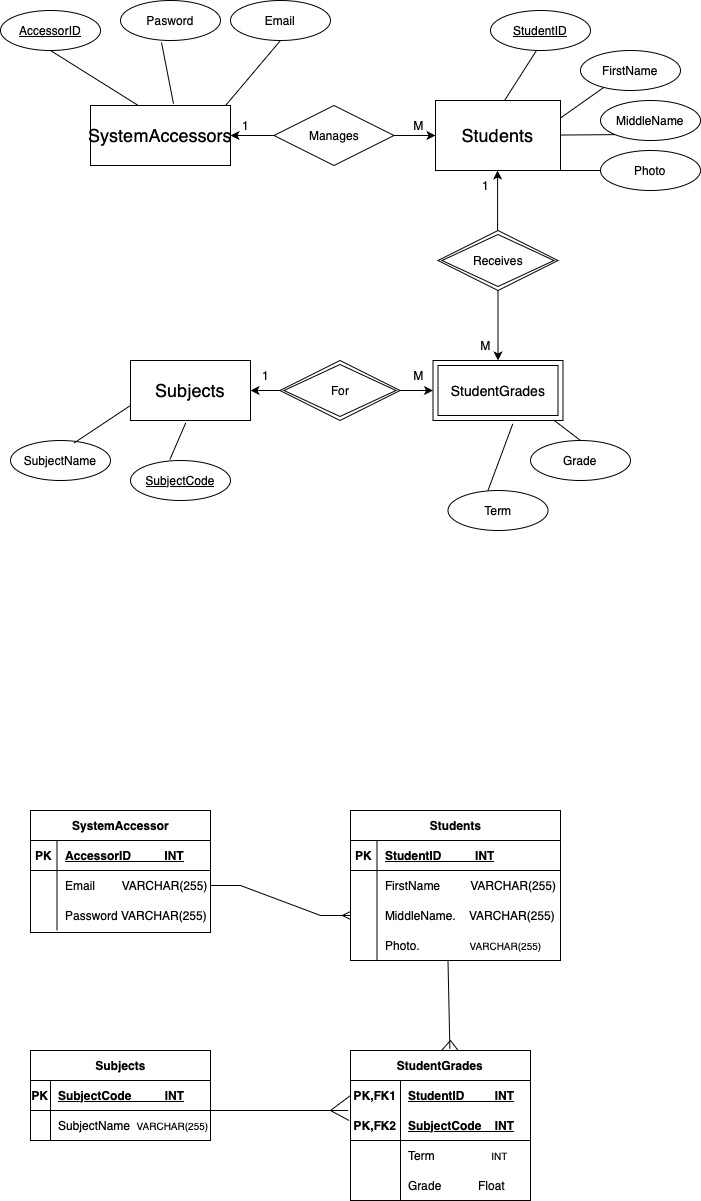
***Entity-Relationship Diagram (ERD)***

An Entity-Relationship Diagram (ERD) shows the various relationships between entities within the system. The database follows a one-to-many relationship structure. One system accessor can manage many students, and one can enroll in many subjects with their grades. The grades table is a weak entity dependent on students and subjects. The primary key for StudentGrades is a composite key consisting of StudentID, Subject Code, and term, which prevents each student from receiving a unique grade for each subject throughout the term. The relationship is structured enough to track student performance efficiently while maintaining referential integrity.



***Relational Model***

The relational model Further develops the ERD into how the data will be structured across separate (normalized) tables in a new table. It keeps the authentication credentials in a system accessors table so only authorized users can access/modify the student records. The student's table contains unique information about the students, including their names and identification numbers. The subjects table helps organize subjects without duplicates, whereas the grades table creates the student-subject relationship in a normalized way with historical performance data. As a result, this relational model improves the accessibility of organized data for retrieval and management.



***Normalization Process***

Normalization is used to store the proper records table-wise. The Student Performance Management System database follows the third standard form (3NF) for appropriate structure and consistency.

*Unnormalized Form (UNF)*

The initial database had repeating groups in a single field, as seen in the image above, with more than one subject, and the corresponding grades were logged under the same row. The underlying structure made it harder to get grades for individual subjects and impossible to query efficiently.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| StudentID | FirstName | LastName | Term | Subjects | Grades |
| 101 | Alice | Johnson | 1 | Mathematics, Science | 85.5, 78.0 |
| 101 | Alice | Johnson | 2 | History, Computer Science | 88.1, 90.5 |
| 102 | David | Smith | 1 | Mathematics, English | 79.4, 75.6 |
| 103 | Emma | Brown | 3 | English, Science | 90.7, 85.7 |

This format resulted in inconsistencies and unnecessary duplication, making it unsuitable for relational databases.

*First Normal Form (1NF)*

To achieve First Normal Form (1NF), repeating groups were removed by ensuring that each field contained a single atomic value. Subjects and grades were separated into individual rows, ensuring that each column held unique data.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| StudentID | FirstName | LastName | Term | SubjectCode | SubjectName | Grade |
| 101 | Alice | Johnson | 1 | MTH101 | Mathematics | 85.5 |
| 101 | Alice | Johnson | 1 | SCI102 | Science | 78 |
| 101 | Alice | Johnson | 2 | HIS104 | History | 88.1 |
| 101 | Alice | Johnson | 2 | CSC105 | Computer Science | 90.5 |
| 102 | David | Smith | 3 | MTH101 | Mathematics | 79.4 |
| 102 | David | Smith | 2 | ENG103 | English | 75.6 |
| 103 | Emma | Brown | 1 | ENG103 | English | 90.7 |
| 103 | Emma | Brown | 3 | SCI102 | Science | 85.7 |

At this stage, each data field contains only atomic values, but partial dependencies still exist, violating Second Normal Form (2NF).

*Second Normal Form (2NF)*

To achieve Second Normal Form (2NF), partial dependencies were removed by dividing the data into separate tables for Students, Subjects, and StudentGrades. Each table had a primary key that uniquely identified its records, ensuring that non-key attributes were fully dependent on the entire primary key.

*Students Table:*

|  |  |  |
| --- | --- | --- |
| StudentID | FirstName | LastName |
| 101 | Alice | Johnson |
| 102 | David | Smith |
| 103 | Emma | Brown |

*Subjects Table:*

|  |  |
| --- | --- |
| SubjectCode | SubjectName |
| MTH101 | Mathematics |
| SCI102 | Science |
| ENG103 | English |
| HIS104 | History |
| CSC105 | Computer Science |

*Grades Table:*

|  |  |  |  |
| --- | --- | --- | --- |
| StudentID | SubjectCode | Term | Grade |
| 101 | MTH101 | 1 | 85.5 |
| 101 | SCI102 | 1 | 78 |
| 101 | HIS104 | 2 | 88.1 |
| 101 | CSC105 | 3 | 90.5 |
| 102 | MTH101 | 2 | 79.4 |
| 102 | ENG103 | 1 | 75.6 |
| 103 | ENG103 | 1 | 90.7 |
| 103 | SCI102 | 3 | 85.7 |

At this stage, each attribute depended on the full primary key, reducing redundancy. However, transitive dependencies still existed, violating Third Normal Form (3NF).

*Third Normal Form (3NF)*

To achieve Third Normal Form (3NF), transitive dependencies were removed by ensuring that every attribute was directly dependent on the primary key. This final step optimized the database structure, improving efficiency and data retrieval.

*System Accessors Table:*

|  |  |  |
| --- | --- | --- |
| AccessorID | Email | Password |
| 1 | [admin@school.com](mailto:admin@school.com) | securePass123 |

*Students Table:*

|  |  |  |
| --- | --- | --- |
| StudentID | FirstName | LastName |
| 101 | Alice | Johnson |
| 102 | David | Smith |
| 103 | Emma | Brown |

*Subjects Table:*

|  |  |
| --- | --- |
| SubjectCode | SubjectName |
| MTH101 | Mathematics |
| SCI102 | Science |
| ENG103 | English |
| HIS104 | History |
| CSC105 | Computer Science |

*StudentGrades Table (Weak Entity with Composite Primary Key):*

|  |  |  |  |
| --- | --- | --- | --- |
| StudentID | SubjectCode | Term | Grade |
| 101 | MTH101 | 1 | 85.5 |
| 101 | SCI102 | 1 | 78 |
| 101 | HIS104 | 2 | 88.1 |
| 101 | CSC105 | 2 | 90.5 |
| 102 | MTH101 | 3 | 79.4 |
| 102 | ENG103 | 1 | 75.6 |
| 103 | ENG103 | 3 | 90.7 |
| 103 | SCI102 | 3 | 85.7 |

3rd Normal form (3NF): We can have the final database structure. Since we already know what normalization is, the final database structure is to achieve the purpose of having our data very well structured without redundancy and the queries we make executed in a timely manner. These principles of normalization make this the perfect streamlined database for everyday use, integrated with the graphical user interface.

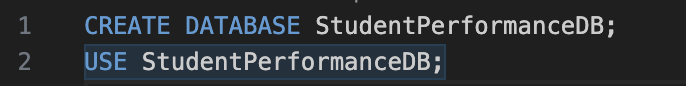
The student performance management system has now achieved the Third Normal Form (3NF) by implementing database normalization. Prevention of Data Redundancy This guarantees that the database is optimized, eradicating duplicates and maintaining data integrity whilst upholding optimized storage and speedy retrieval. In the subsequent phase, Physical Design, SQL scripts implementing these relationships get executed, impacting the MySQL database.

3.1.4 The Physical Design

The Physical Design phase involves implementing the database schema by translating the conceptual and logical models into SQL code. This step ensures that all relationships, constraints, and indexing strategies are properly structured to enhance database performance, enforce data integrity, and support efficient querying.

*Creating the Database:*

The following SQL command initializes the Student Performance Management System database:



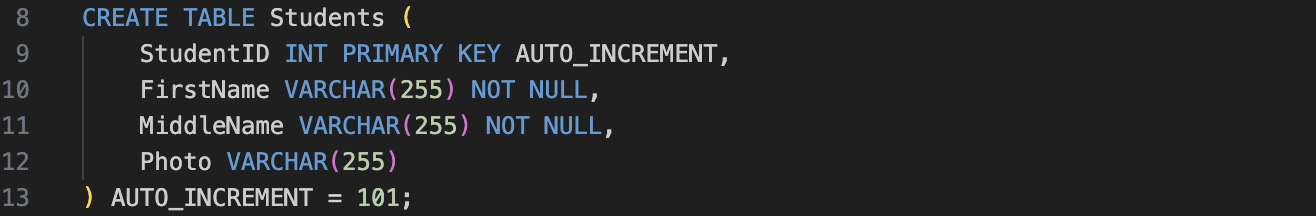
*Creating the System Accessors Table:*

This table stores administrator login credentials and ensures that only authorized users can access and modify student performance data.



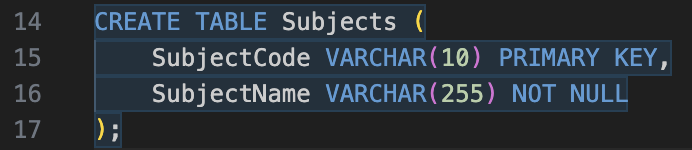
*Creating the Students Table:*

The Students table holds essential information about each student, ensuring that every student has a unique enrolment number.



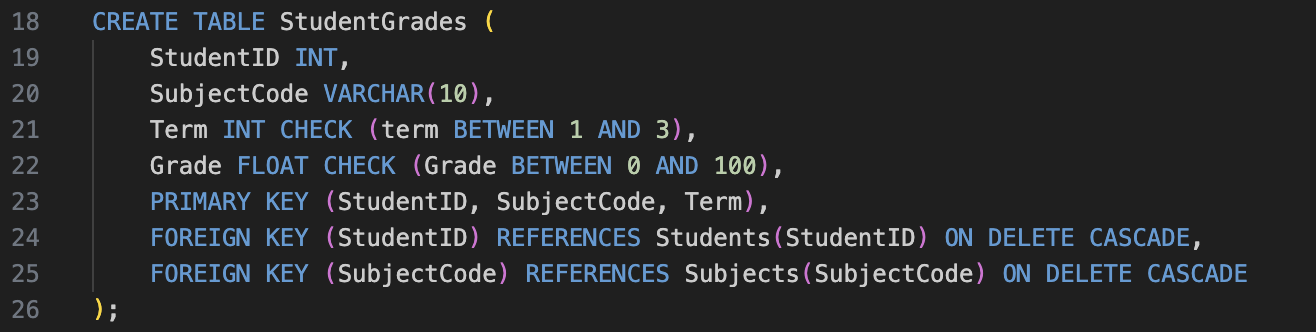
*Creating the Subjects Table:*

Each subject is identified using a unique subject code and subject name.



*Creating the StudentGrades Table (Weak Entity with Composite Primary Key):*

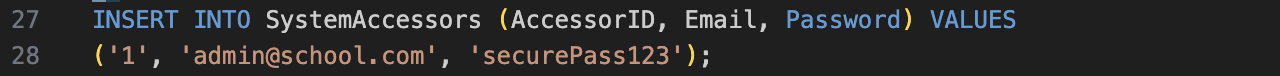
The Grades table stores students’ performance data, linking them with their respective subjects and terms.



To populate the database, realistic sample data is inserted. These entries will be used for testing the system’s functionality and performance.

*Inserting Sample System Accessor:*

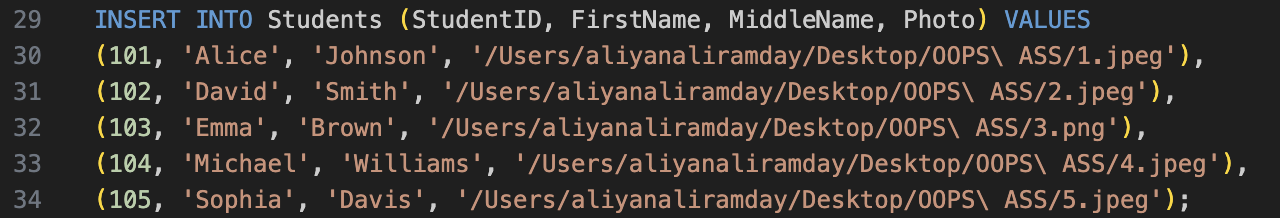
The system starts with one administrator responsible for managing student records.

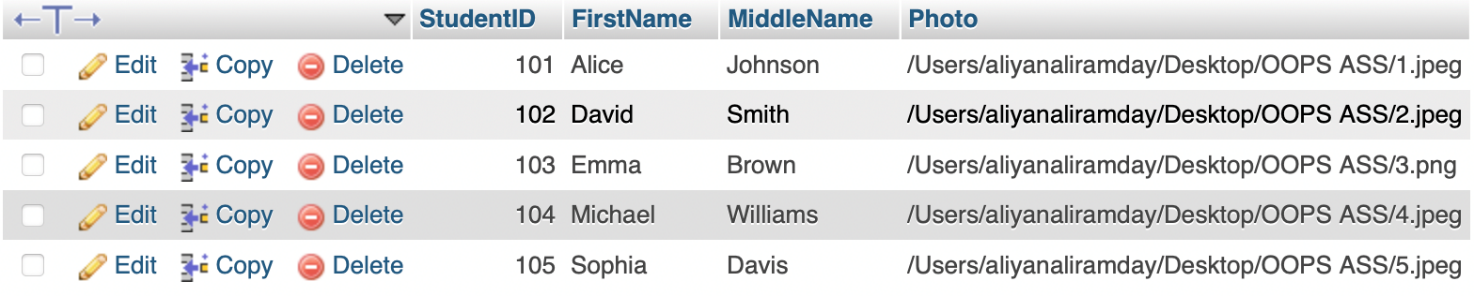




*Inserting Sample Students:*

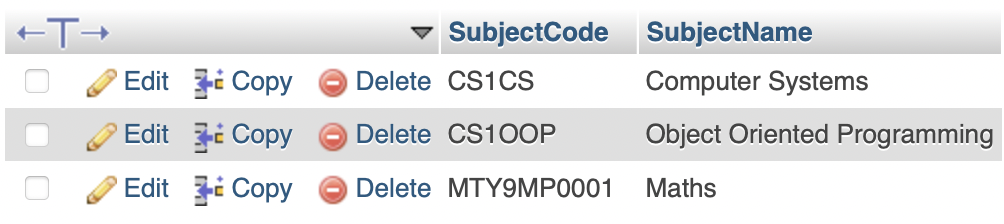
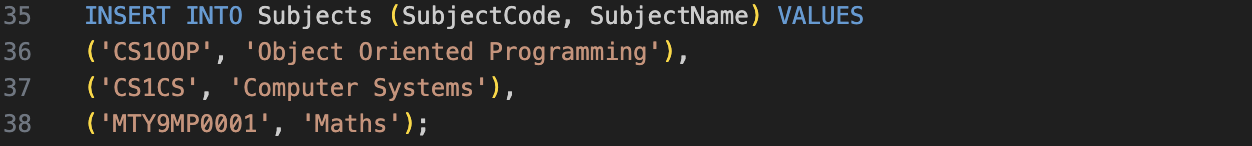
A list of five students is added to the database.



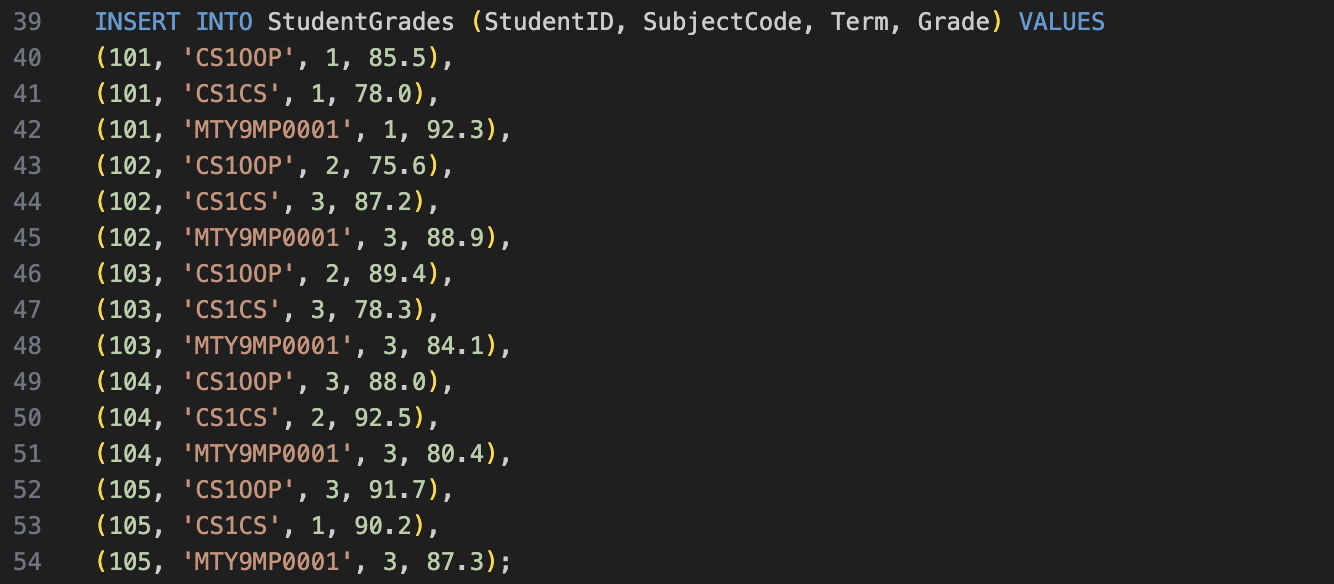


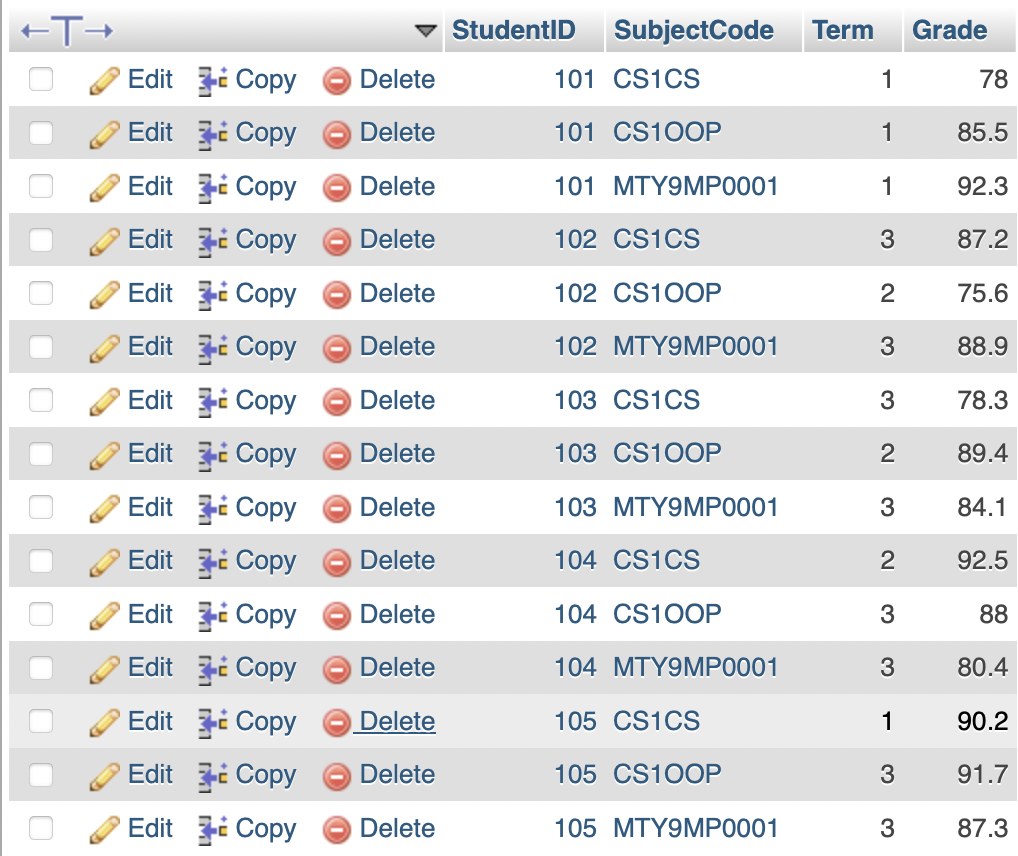
*Inserting Sample Subjects:*

Subjects have been updated with realistic course codes.

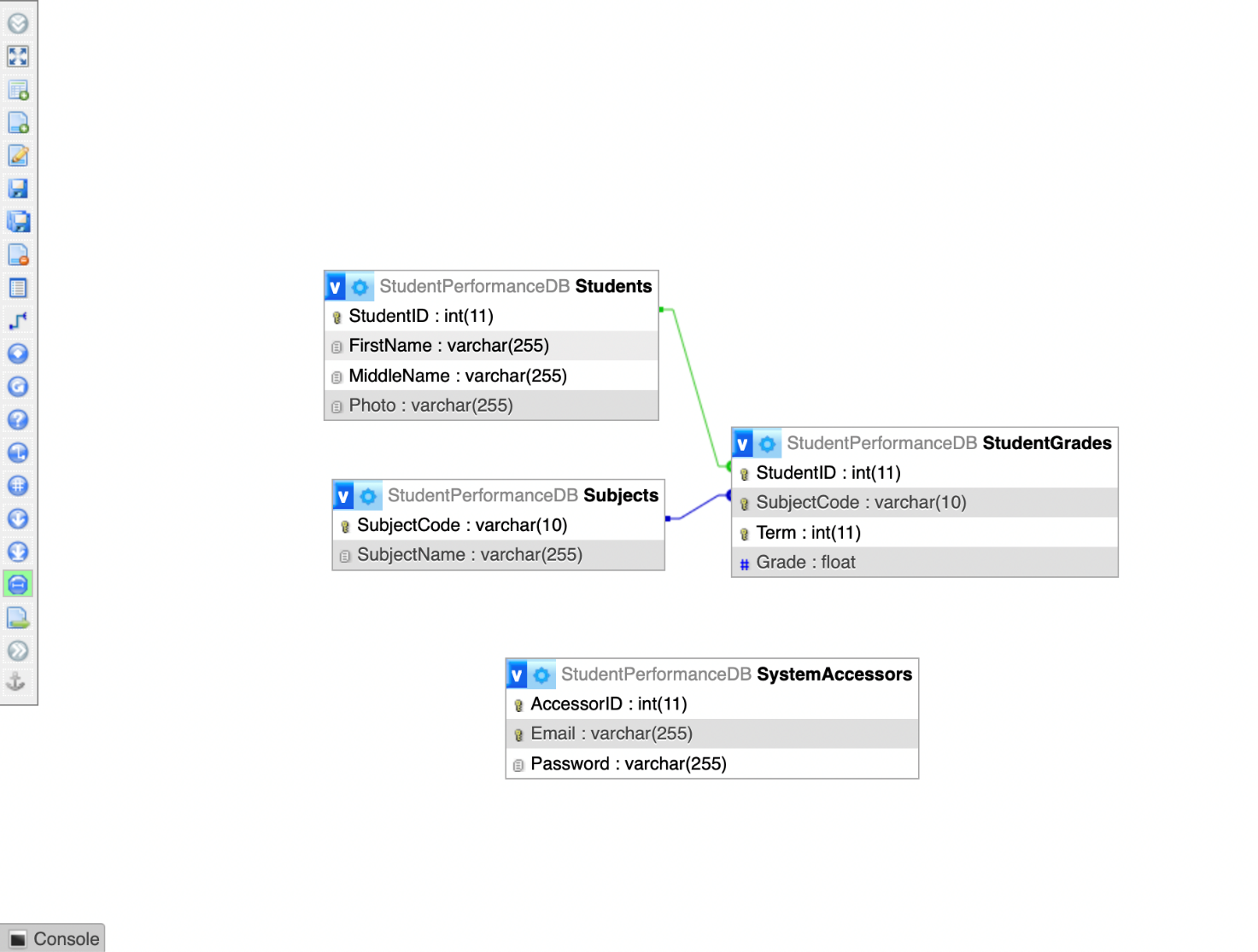


*Inserting Sample Grades:*

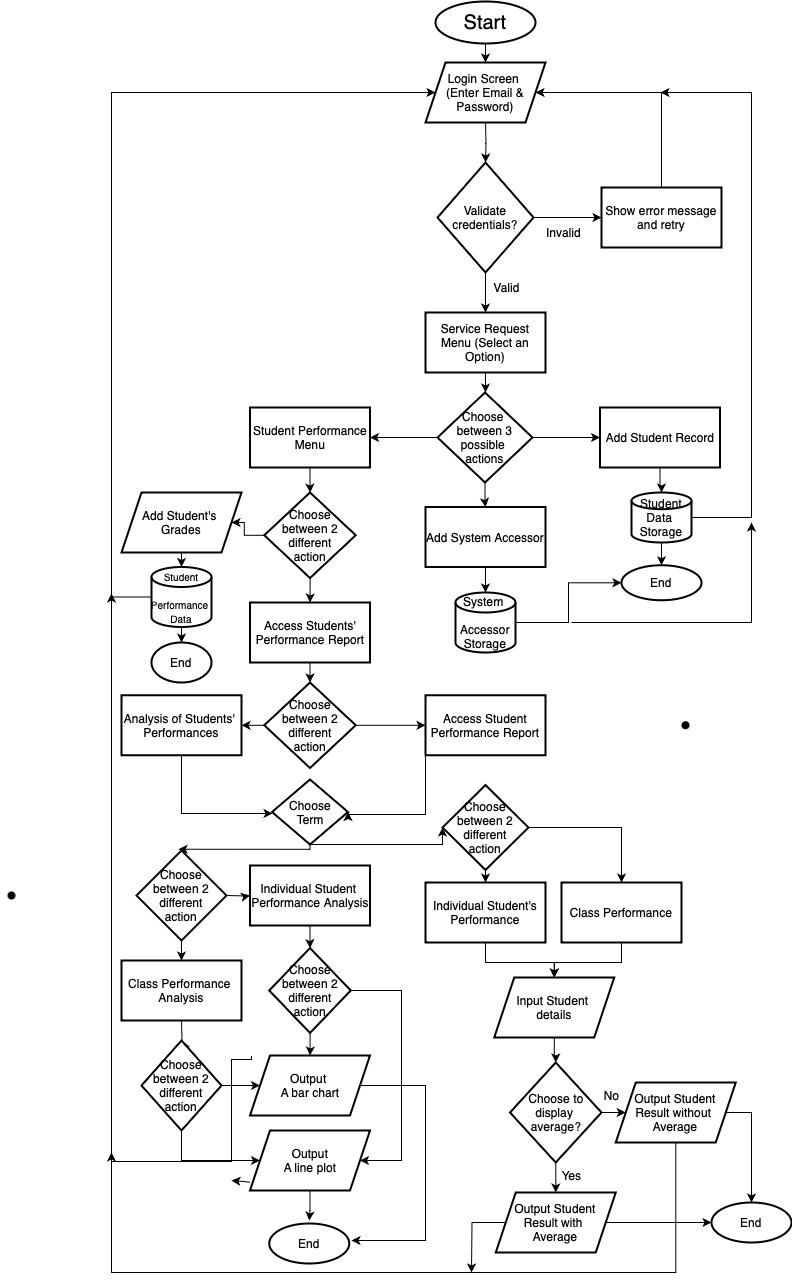
Students' performance records are stored term-wise, with randomized grade values for realistic data representation.  




We developed the MySQL database using Visual Studio Code and used XAMPP to run a local MySQL server for execution. I wrote and tested SQL queries in VS Code but viewed the database tables, designed relationships, and checked outputs in XAMPP-accessible phpMyAdmin. This gives programmers opportunities for flexible development  without the need for an overview of the database.



## 3.2 The flowchart for Program Design



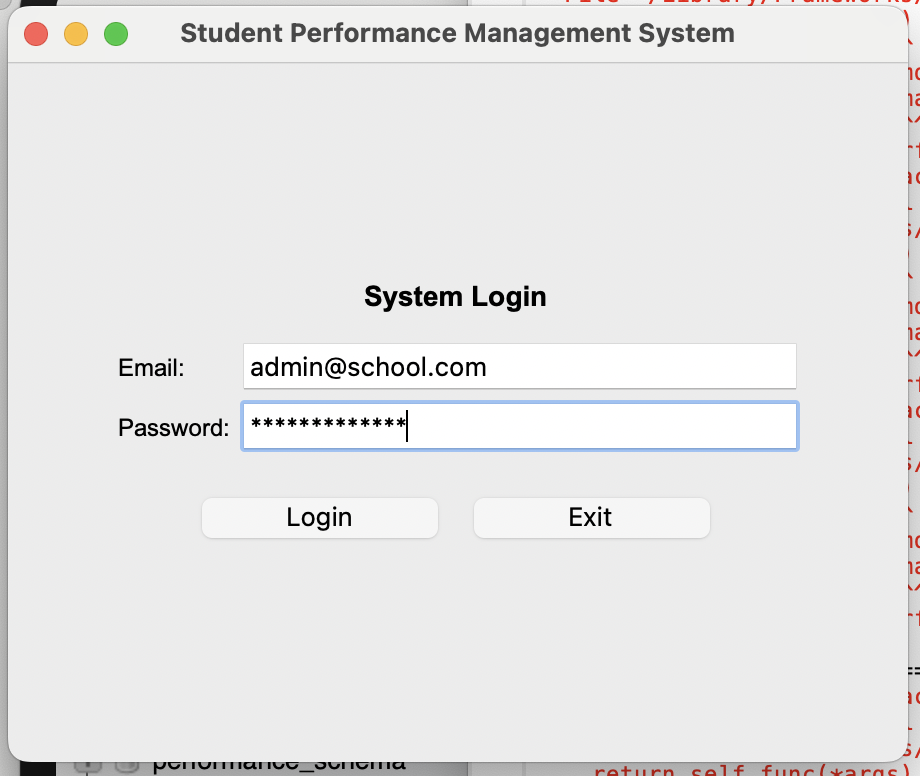
# 4. Testing and Implementation

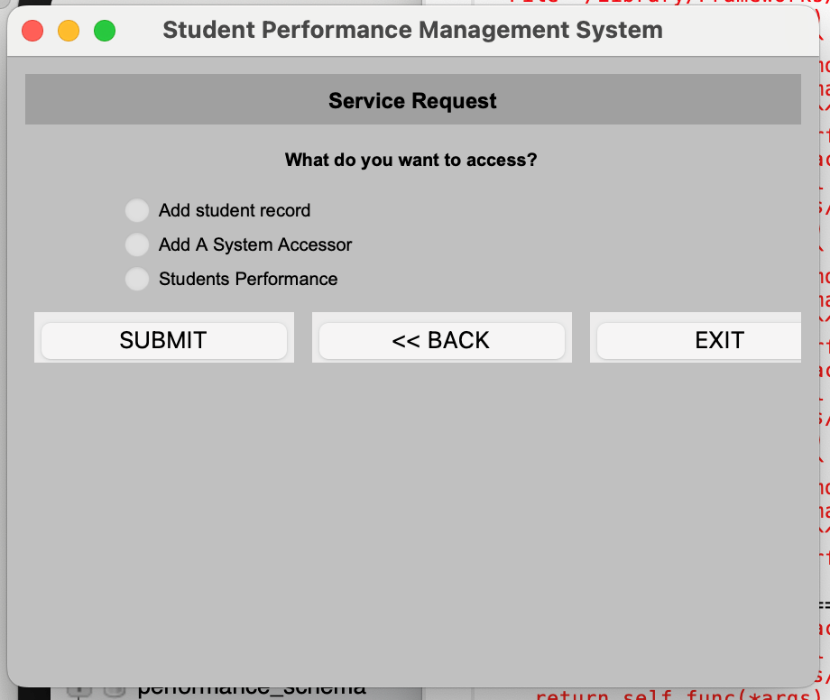
The Student Performance Management System is implemented in this study in an overall system testing plan to confirm the functionality, usability, and robustness of the system in this study. The testing process involved multiple levels of validation to identify and address any issues before deployment. It included unit testing, integration testing, regression testing, user acceptance testing (UAT), and system testing. This document covers project testing planning, test data analysis, and GUI verification.

It was tested in isolation so we could test only the system's core function. The test was targeted at ensuring individual components worked as intended through different test cases.

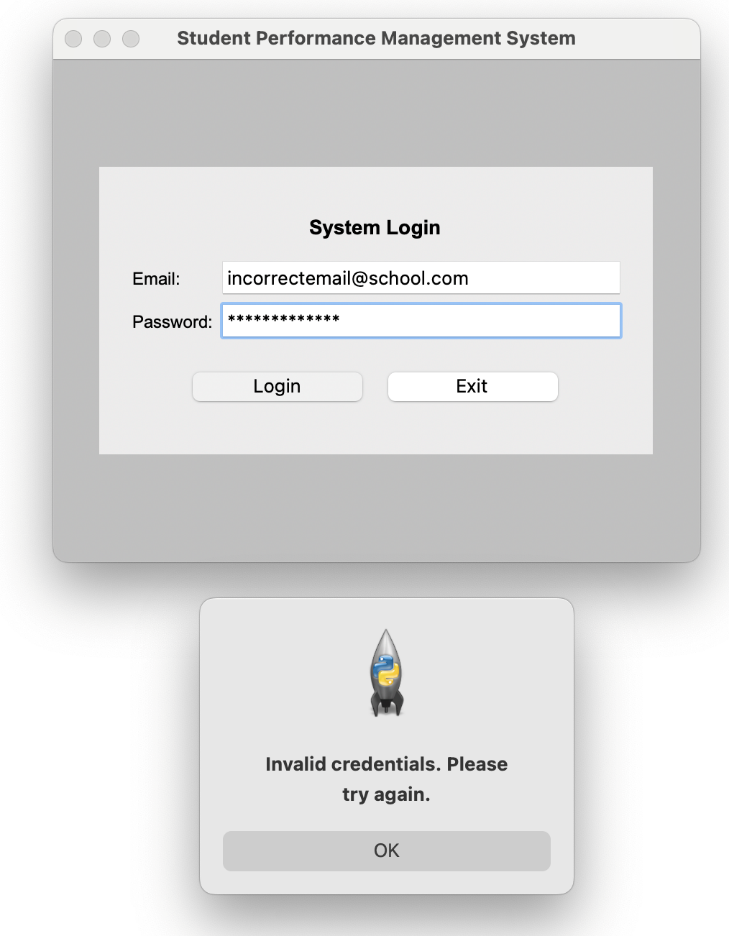
Based on the above strategies, one of the areas tested was Login Authentication, where valid and invalid credentials were entered into the system to check whether they correctly verified user access. When the correct email and password were entered, the user was successfully authenticated with a correct response, and when incorrect, an error message was displayed. Validation messages also appeared when empty fields were submitted with an attempt to log in, preventing users from leaving any entry blank.

**Correct email and password inputs successfully authenticated the user**

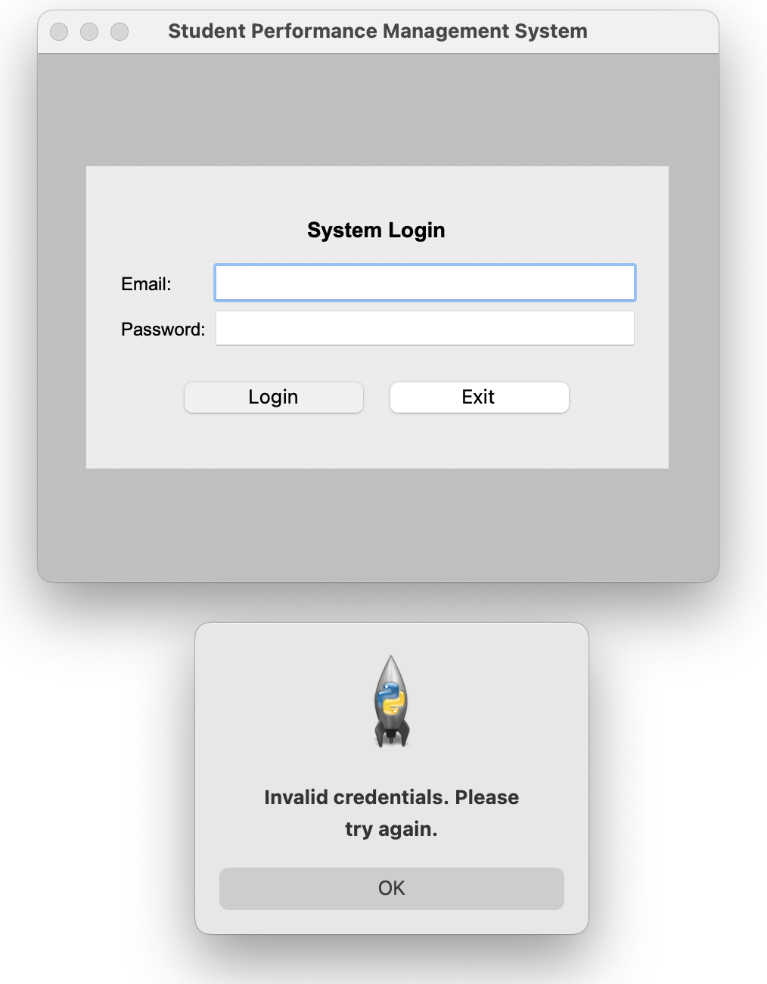




**Incorrect credentials prompted an error message**



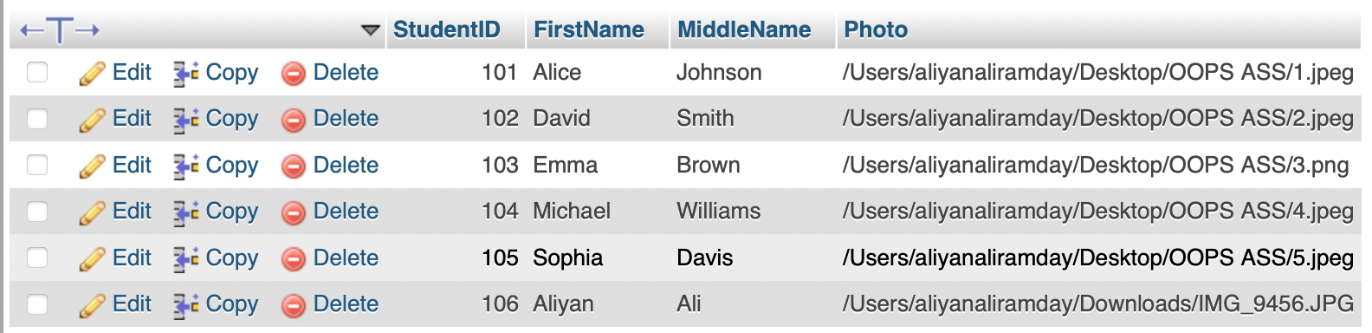
**Attempts to log in with empty fields triggered validation messages**



In the Student Record Management module, tests were performed to verify the accurate addition, updating, and retrieval of student records. Test cases included adding a student with all necessary information, leaving out required fields to observe error handling, and retrieving student data to ensure consistency with the database.

**Successfully adding a student with all necessary information**

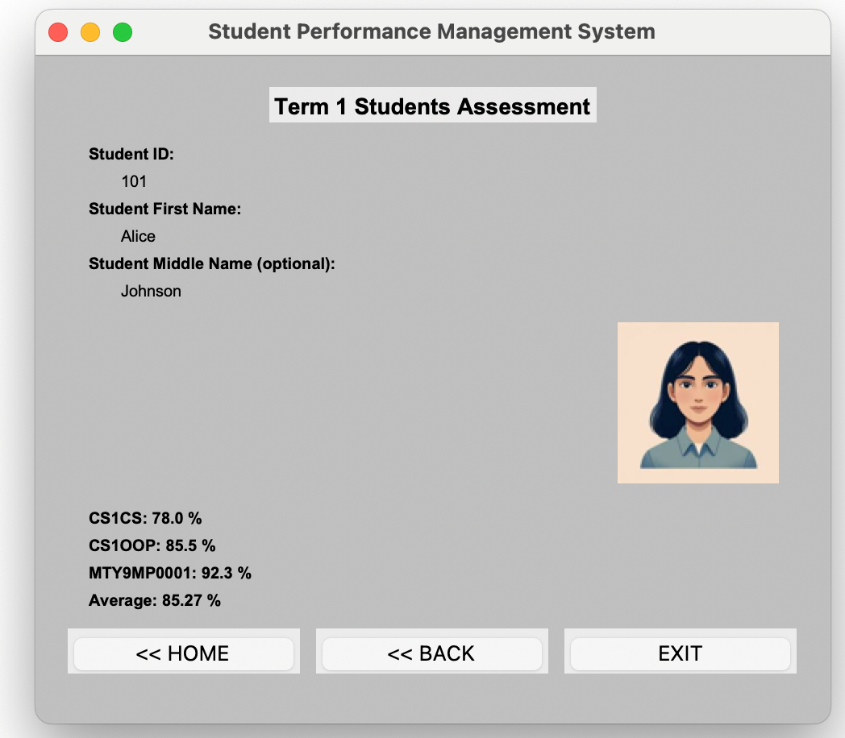




**Leaving out required fields to observe error handling**

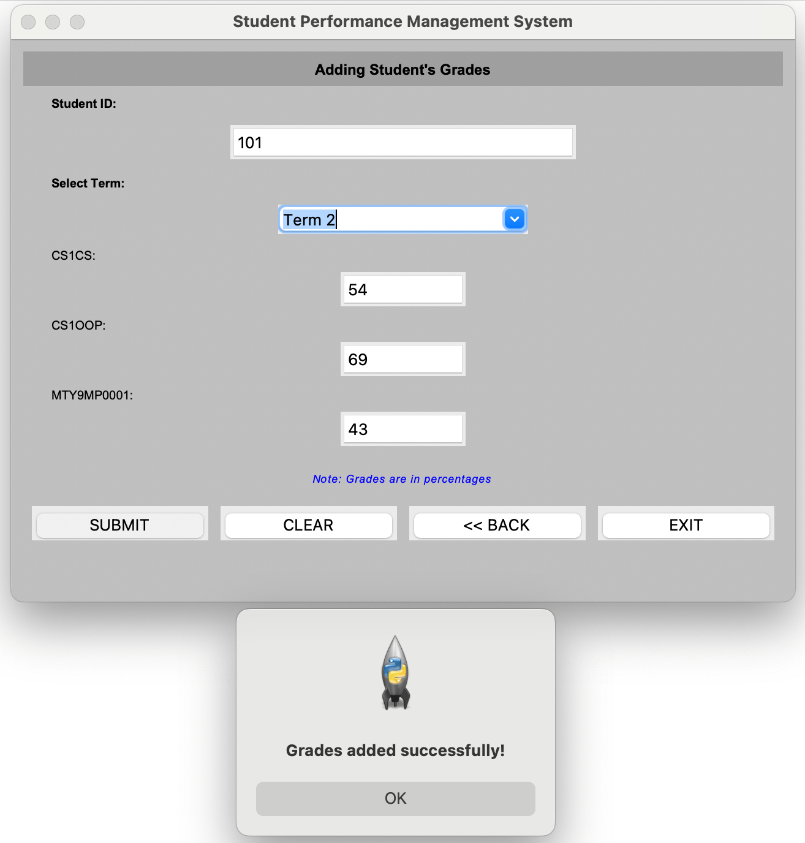


**Retrieving student data to ensure consistency with the database**

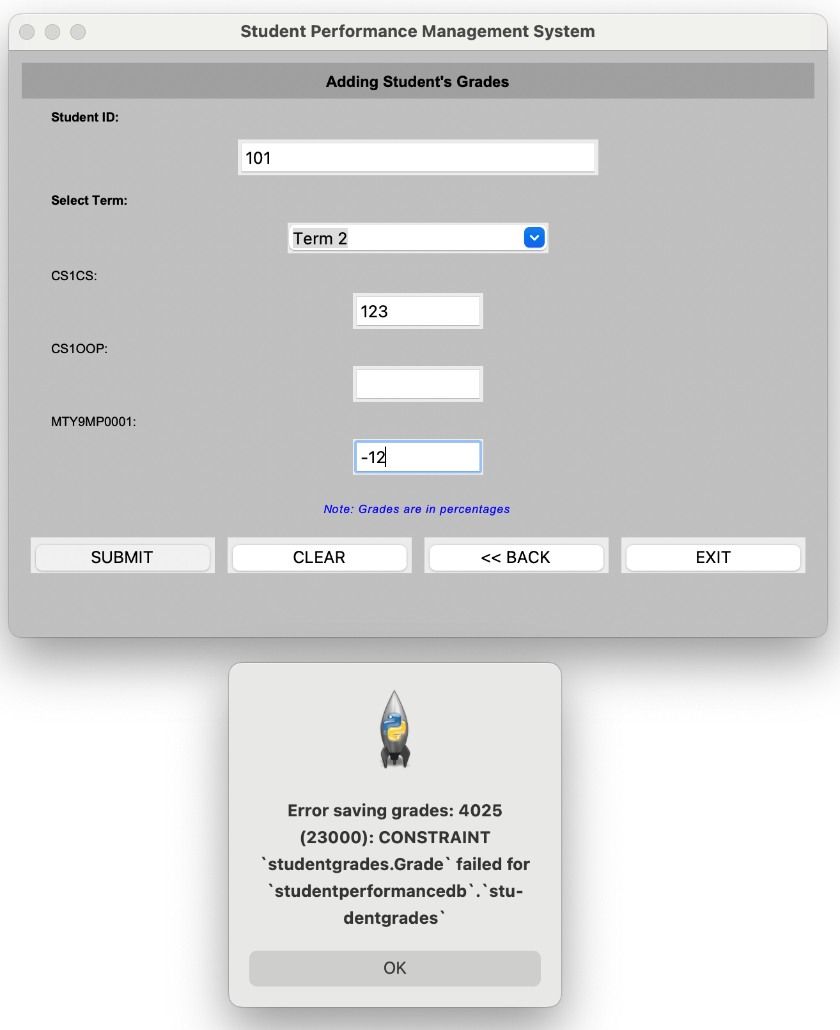


The Grade Entry System was tested by inputting various grades, including valid entries within the 0-100 range, invalid grades outside this range, and empty fields. The system correctly stored valid entries and rejected invalid ones.

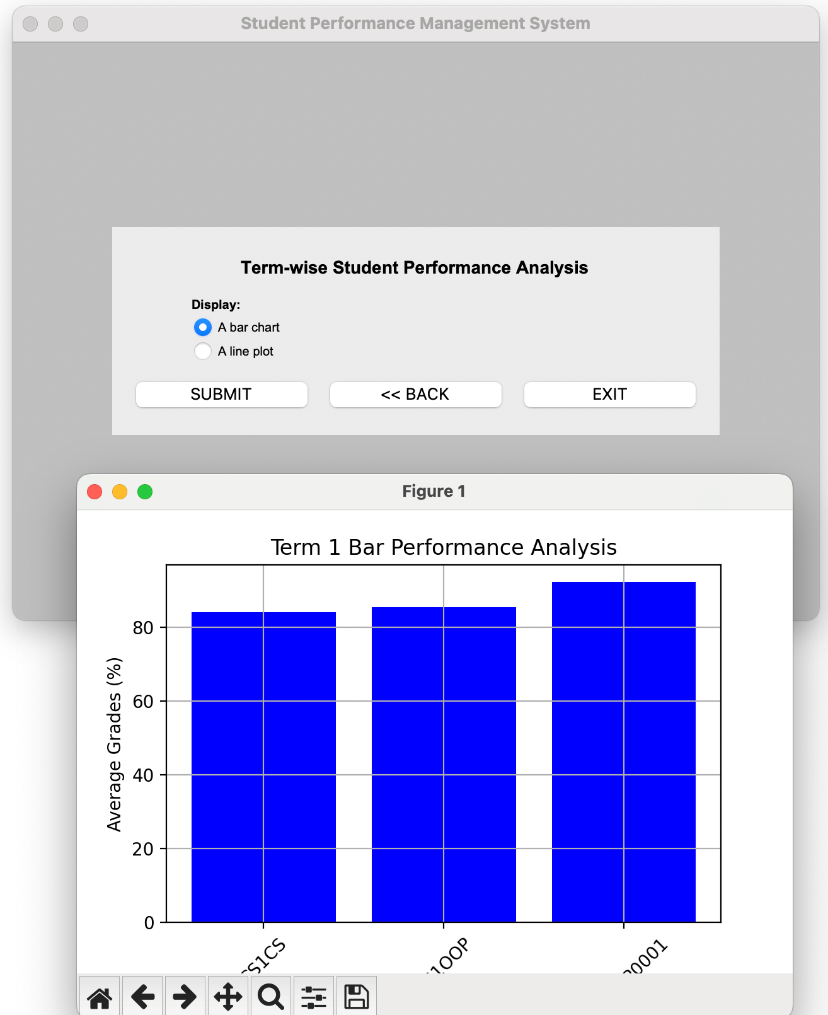
**The system correctly stored valid grade entries**

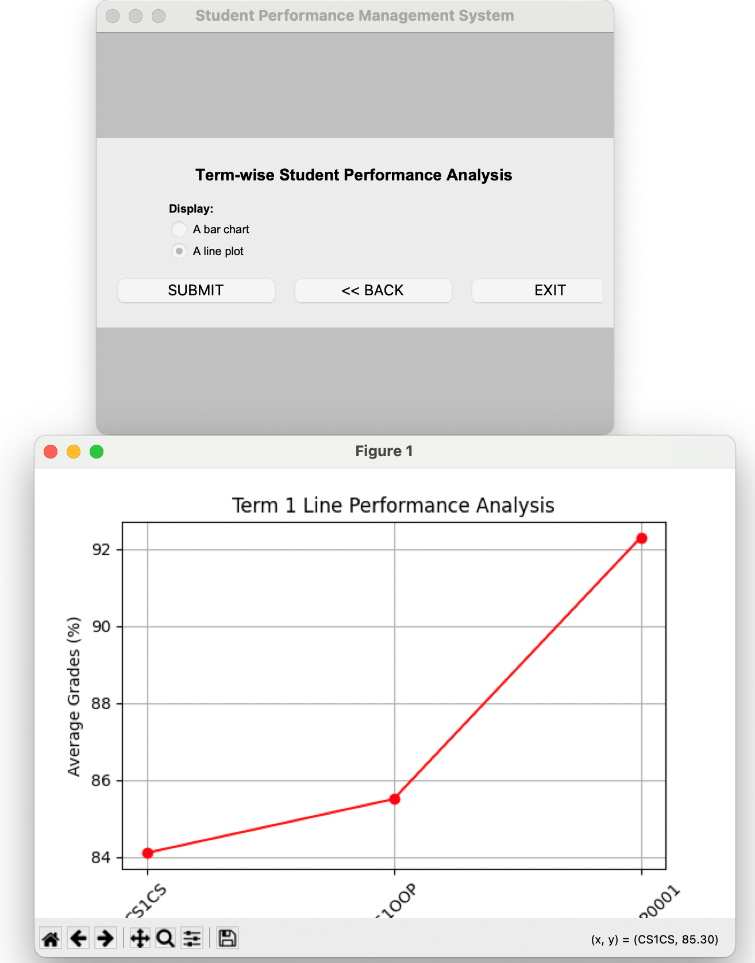


**The system correctly rejected invalid grade entries**



For the Performance Analysis Module, tests ensured that the system generated the correct performance graphs based on student grades.





# 5. Critical Discussion & Recommendations

Database interaction, managing user error with exceptions, and developing a responsive GUI were some of the many features of the Student Performance Management System, thus adding a few challenges during the implementation stage. This was one of the biggest challenges; implementing validations against the database had to be set up and appropriately channelized. The current version has a potential security hole because it does not hash the passwords. Solve that on your next release by using hashed passwords and checking that against the user's input.

The nice thing else was how the student image file upload was handled. The subsystem accepts image files without checking their type, which can result in a corrupted database when unsupported files are uploaded. Adding support for PNG would help by validating data only for the right image formats (jpg, obviously).

Note that the grade entry module works currently, but it does not check that a student ID provided to it is in the database before it inserts grades. If an invalid student ID is used, this can cause orphaned records. Instead of just doing an addition, we could pass through some validation before allowing grades to be saved.

The performance analysis requires extracting and displaying the student performance; it is one of the modules. However, this module does not validate whether the required term is present in the database before producing reports using MATPLOTLIB. Choosing the wrong term will yield an error or an empty graph. Tighter checks on data before searching the database and more informative error messages are all good UX.

We are dealing with the challenges notwithstanding, as the system is a systematic approach to allowing teachers to have a tidier user interface to keep a record of their students. Teachers can see how students trend across different terms. As an educator, I realize that it is not practical to scroll through a list of all the individual students and that looking up student records in real time would be very beneficial in a GUI.

Potential improvement areas include system-optimized performance, security features, more robust error handling, etc. A new line of improvement could be an automated backup of student data to increase reliability and safety in case of an unintentional failure.