National Economics University

Faculty of Data Science and Artificial Intelligence

Course: Introduction to Databases

Student Information Manager

Database Normalization, MySQL Schema, and GUI Application

Student Information:

1. Nguyen Van Tue — 11247366
2. Pham Huy Thanh — 11247351
3. Le Duy Quyen — 11247345

Class: AI66B

Instructor: Dr. Tran Hung

December 12, 2025

Contents

List of Figures

List of Tables

Abstract

This report presents the design and full-stack implementation of the Student Information Manager, a relational database system developed to improve the accuracy, efficiency, and scalability of academic data administration. The project follows a structured Software Development Life Cycle (SDLC), starting from conceptual modeling and normalization, advancing through database construction in MySQL, and culminating in application-layer integration and analytics.

The system is engineered from raw, unorganized datasets and transformed through rigorous normalization procedures to achieve Third Normal Form (3NF), thereby eliminating redundancy, mitigating update anomalies, and reinforcing referential integrity. A comprehensive Entity–Relationship Diagram (ERD) and an optimized logical schema were constructed to ensure clear representation of relationships across core entities such as Students, Lecturers, Subjects, Classes, and Enrollments. The final schema incorporates composite primary keys, foreign key cascades, domain constraints, CHECK validations, ENUM types, procedural data generation, and performance-oriented indexing.

On top of the database layer, a modular Python backend was implemented using a clean architecture pattern. This includes standardized model classes, raw SQL with parameterized queries for security, a dedicated service layer for business logic enforcement, and a robust connection handler using environment-based configuration. These components collectively support stable and secure CRUD operations while preventing common issues such as SQL injection, duplicate enrollments, and inconsistent updates.

A user-friendly Graphical User Interface (GUI) complements the backend, enabling intuitive execution of CRUD workflows and enhancing interaction between end-users and the underlying relational system. Additionally, analytical components were incorporated to generate dynamic dashboards, key performance indicators (KPIs), and complex SQL-driven insights for academic decision support.

Overall, the evaluation demonstrates that the system achieves strong data consistency, optimized query performance, and operational scalability. The project showcases a cohesive end-to-end solution integrating database theory, software engineering principles, and practical system implementation, illustrating how well-designed data infrastructures can tangibly improve educational information management.

Acknowledgments

We would like to express our sincere appreciation to our course instructor for their exceptional guidance, rigorous academic standards, and continuous support throughout the development of this project. Their expertise in database systems, normalization theory, and software engineering provided the foundational knowledge that enabled us to approach this project with both technical depth and methodological precision. The instructor's constructive feedback was instrumental in helping us refine our data model, strengthen our system architecture, and uphold industry-level best practices.

We also extend our gratitude to all team members for their unwavering commitment and collaborative spirit. Each member played an essential role in transforming a raw conceptual idea into a fully functional and well-engineered Student Information Manager system. From conceptual modeling (ERD, functional dependency analysis, and normalization through 3NF) to the implementation of a robust MySQL schema, optimized seeding procedures, and a Python-based backend with structured models and CRUD operations, this project is the result of sustained teamwork and shared dedication. The integration of a user-friendly GUI and analytics dashboards further highlights the collective effort and technical synergy of the group.

This project would not have been possible without the combined knowledge, persistence, and cooperation of everyone involved. We are grateful for the opportunity to engage in this meaningful learning experience and for the academic environment that encouraged us to think critically, design systematically, and build with purpose.

1 Introduction

1.1 Problem Context

Modern higher education institutions rely heavily on accurate, efficient, and scalable information systems to manage student records, course offerings, lecturer assignments, and academic results. However, when academic data is initially collected, it frequently appears in unnormalized, semi-structured formats that combine multiple entities into single records and contain non-atomic or repeating attributes. Such disorganized data structures introduce classical integrity problems, including redundancy, insertion anomalies, update anomalies, and deletion anomalies. Over time, these issues not only compromise data reliability but also hinder performance in transactional and analytical workloads, making the system difficult to maintain and scale.

To mitigate these challenges, the development of a rigorously normalized relational database is essential. This project centers on transforming a raw UNF dataset into a logically consistent and semantically sound schema that satisfies Third Normal Form (3NF). Through systematic decomposition guided by functional dependencies, the final design eliminates unnecessary duplication, preserves referential integrity, and provides an optimal foundation for query processing. The completed database serves as the underlying infrastructure for storing student information, managing course structures, supporting lecturer assignments, and recording enrollment outcomes in a stable and high-integrity environment.

1.2 Project Scope

This report documents the complete conceptual, logical, and implementation workflow for constructing a university-scale Student Information Management System. The scope of work includes:

- A full normalization pipeline from UNF $\rightarrow$ 1NF $\rightarrow$ 2NF $\rightarrow$ 3NF, supported by formal functional dependency (FD) derivation and theoretical justification.
- Development of a fully normalized relational schema incorporating primary keys, foreign keys, cardinalities, participation constraints, and ER-to-relational mapping principles.
- Construction of a backend data layer using Python (mysql-connector) with a clean, modular architecture (models, services, connection management), supporting robust CRUD operations and safe parameterized SQL execution.

- Integration of a user-friendly Python GUI application, enabling real-time interactions with the database for administrative tasks such as managing students, lecturers, subjects, classes, and enrollments.
- Implementation of analytical SQL queries and dashboards to visualize academic performance trends, enrollment patterns, and key institutional metrics.
- System evaluation covering performance, data consistency, exception handling, and recommendations for future enhancements such as indexing strategies, API extensions, or migration to a web-based interface.

In summary, this project bridges foundational database theory with practical software engineering to produce a complete, fully functional Student Information Manager. The final system demonstrates structural rigor, operational reliability, and strong alignment with real-world academic data management requirements.

2 Background & Related Work

2.1 Database Normalization

Database normalization is a core methodology in relational database design. Its purpose is to organize data in a way that reduces redundancy, prevents update anomalies and maintains logical consistency across relations. The concept originates from the foundational work of E. F. Codd, whose relational model introduced a rigorous framework for structuring data [1].

Raw datasets typically begin in the Unnormalized Form (UNF), where attributes may contain repeating groups, multi-valued fields or multiple entity concepts mixed into a single record. Converting UNF into the First Normal Form (1NF) introduces atomicity, ensuring that each attribute holds a single, indivisible value. This marks the first step toward a structured relational schema.

The Second Normal Form (2NF) extends this by removing partial dependencies, meaning all non-key attributes must depend on the entire primary key in relations that use composite keys. The Third Normal Form (3NF) further eliminates transitive dependencies so that non-key attributes depend only on the primary key and not on each other. The progression from UNF to 3NF yields a cleaner, more maintainable schema that supports reliable query processing and minimizes structural anomalies. These normal forms were formalized and widely adopted through early relational research [2].

Functional dependencies play a central role in this process by defining the precise relationships among attributes. With well-defined keys and dependencies, designers can validate the correctness of their schema before implementation.

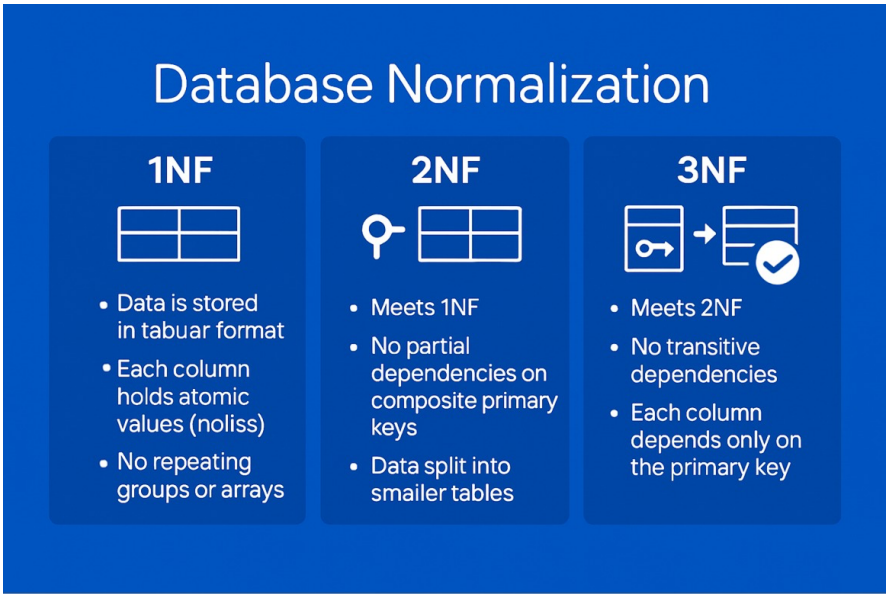

Figure 1: Overview of 1NF, 2NF, and 3NF in the normalization process.

Normal Form	**Definition and Requirements**
UNF	Unstructured data that may contain repeating groups, multi-valued attributes and mixed entity information.
1NF	Requires atomic attribute values and the removal of repeating groups. Establishes the baseline relational structure.
2NF	Requires the relation to be in 1NF and removes partial dependencies, ensuring full functional dependency on the primary key.
3NF	Requires the relation to be in 2NF and removes transitive dependencies so that non-key attributes depend only on the primary key.

Table 1: Comparison of UNF, 1NF, 2NF, and 3NF during database normalization.

2.2 Relational Database Design

Relational databases represent data using relations composed of tuples and attributes, grounded in set theory and predicate logic. Structural constraints such as primary keys and foreign keys preserve entity and referential integrity, ensuring that records remain unique and that relationships between tables remain consistent.

A well-designed relational schema supports clear entity boundaries, efficient indexing and scalable transaction processing. In this project, these principles guided each stage of schema development, from identifying core entities to defining attributes and constraints.

The entity-relationship model serves as the conceptual foundation for translating real-world objects into relational schemas, making it a widely used technique in database design [3].

2.3 Graphical User Interfaces

Graphical User Interfaces (GUIs) provide an accessible layer for interacting with database systems. Instead of relying on direct SQL commands, users interact with visual elements such as buttons, tables and input fields. This reduces user error and improves usability, especially in systems that require frequent data entry and retrieval.

In this project, the GUI acts as the bridge between users and the MySQL backend. It enables CRUD operations for managing students, lecturers, subjects, classes and enrollments, ensuring that the system remains practical for real-world academic administration. Python frameworks such as Tkinter, PyQt and CustomTkinter offer the tools needed to build these interfaces and integrate them with database logic.

The GUI layer demonstrates how the normalized schema and relational design translate into a functional application, validating the effectiveness and robustness of the underlying database model.

3 Methodology

3.1 Normalization

The project begins by examining the dataset in its Unnormalized Form (UNF), where repeating groups, multi-valued attributes, and merged entity information are common. These structural issues are the root of redundancy and update anomalies. Normalization provides a structured method to fix this, following the relational principles introduced by Codd in his foundational work [1].

A full set of Functional Dependencies (FDs) was identified to guide how the dataset should be decomposed. The overall process follows the definitions of normal forms established in relational database literature [4]. Based on these dependencies, the dataset was transformed into the First Normal Form (1NF) by removing repeating groups and ensuring atomic values.

Next, the schema was refined into the Second Normal Form (2NF) by removing partial dependencies on composite keys. The final step, moving the relations into the Third Normal Form (3NF), removes transitive dependencies, ensuring that all non-key attributes depend only on the primary key. These principles align with the guidance provided in established normalization research [2] and textbook standards [5].

The resulting schema is stable, consistent, and avoids the classic anomalies of unnormalized structures.

3.2 Final ERD

After the normalized relations were finalized, they were combined into a complete Entity–Relationship Diagram (ERD). The ERD outlines entities, attributes, primary keys, foreign keys, and relationship constraints. This modeling method follows the classical Entity–Relationship approach introduced by Chen [3].

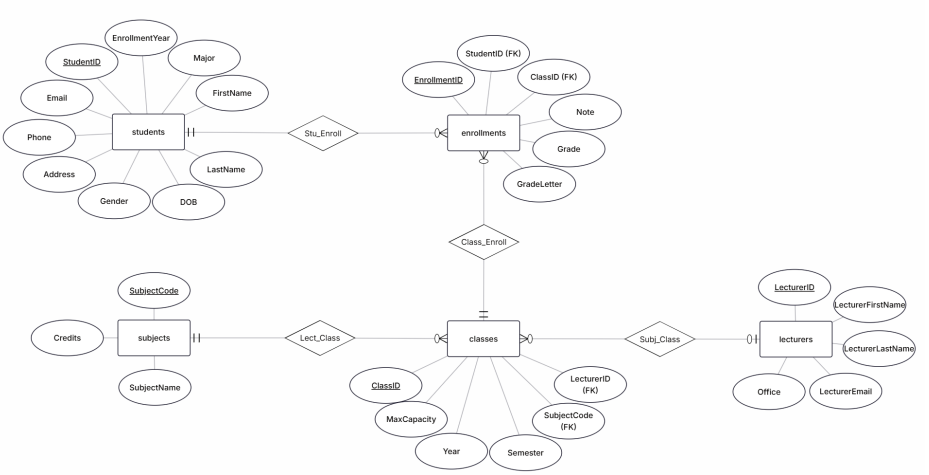

Figure 2: Final ERD representing the 3NF relational structure.

3.3 System Architecture

The software architecture follows a layered design:

- **Presentation Layer:** A GUI interface that allows users to interact with the database.
- **Application Logic Layer:** Handles validation, system logic, and communication between frontend and backend.
- **Data Layer:** Manages SQL queries, connections, constraints, and transactions.

This style of modular design is consistent with widely adopted software engineering practices and design pattern principles discussed in modern development literature [6].

3.4 Database Implementation

The database is implemented through two SQL scripts:

- `schema.sql` — creates tables, data types, relationships, and indexing strategies.
- `seed.sql` — inserts sample data for testing and demonstration.

The implementation follows standard MySQL development guidelines [7] and practical approaches used by database administrators [8].

3.5 Application Workflow

The application's GUI modules support CRUD operations for students, lecturers, subjects, classes, and enrollments. Each workflow includes input validation, error handling, backend SQL operations, and straightforward navigation.

These components demonstrate how a relational database can support real academic management tasks while maintaining consistency and usability.

4 Results

4.1 GUI Screenshots

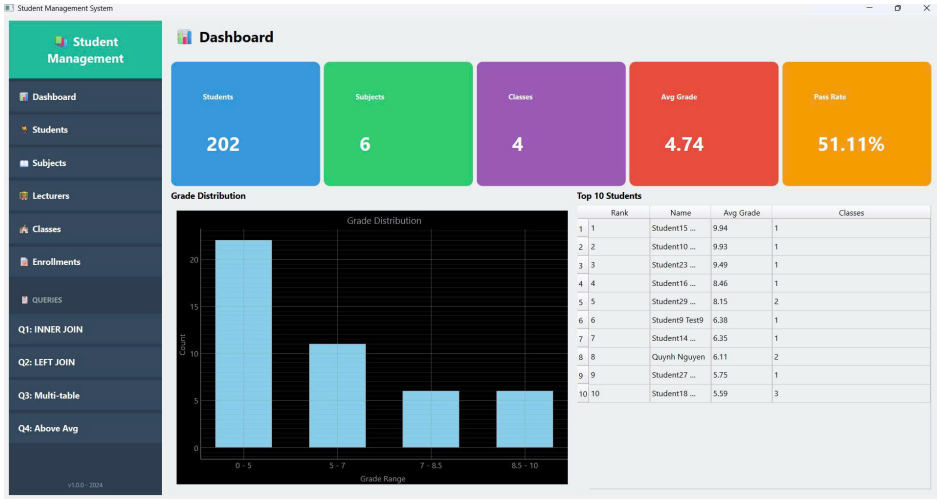

Figure 3: Dashboard Interface

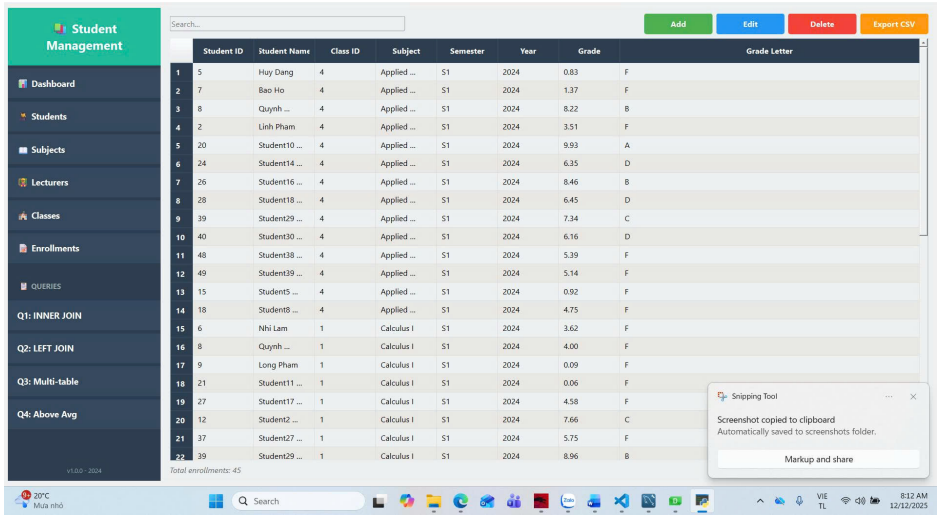

Figure 4: Enrollment Management Screen

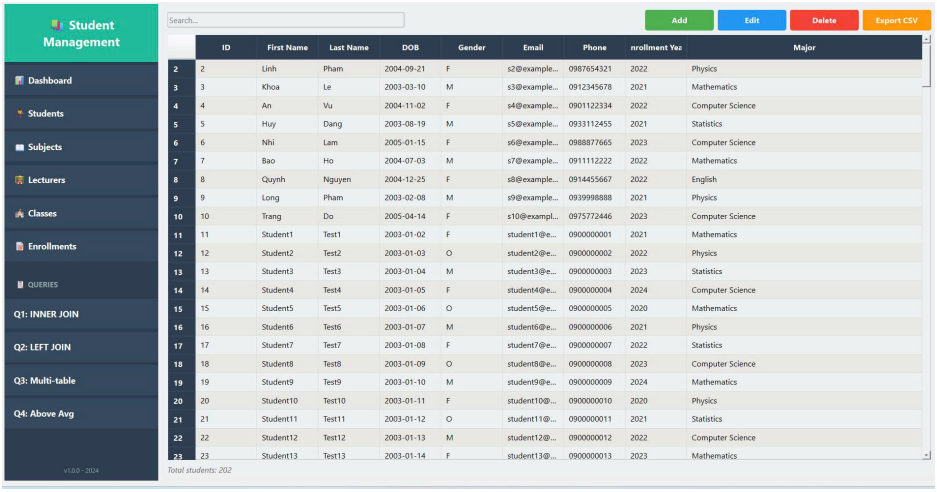

Figure 5: Student Management Interface

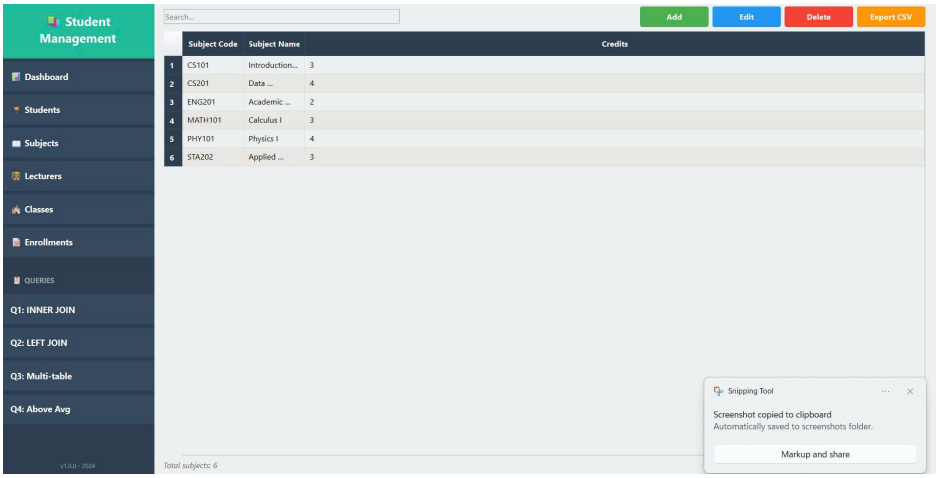

Figure 6: Subject Management Screen

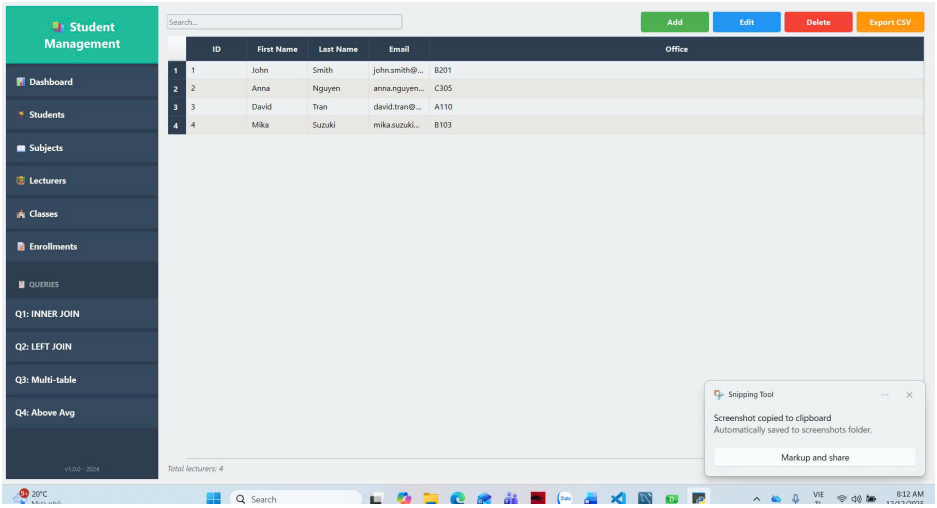

Figure 7: Lecturer Management Screen

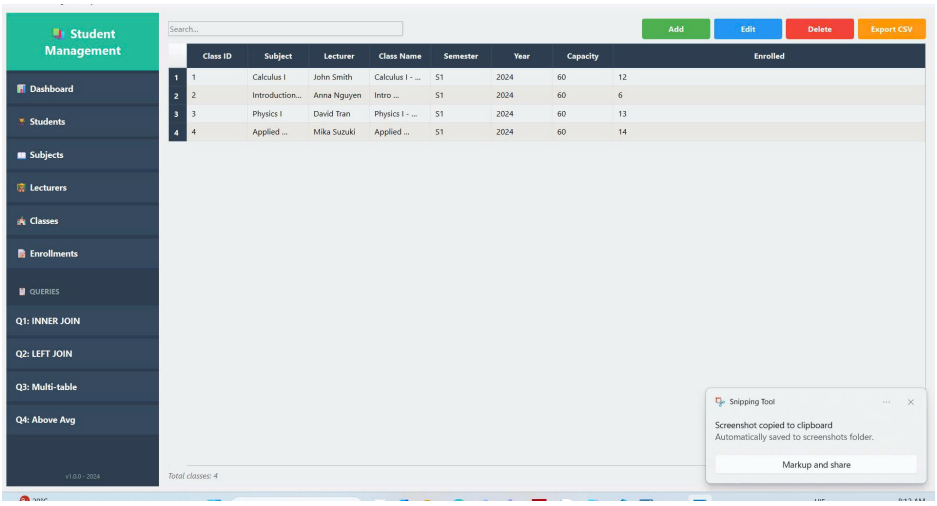

Figure 8: Class Management Interface

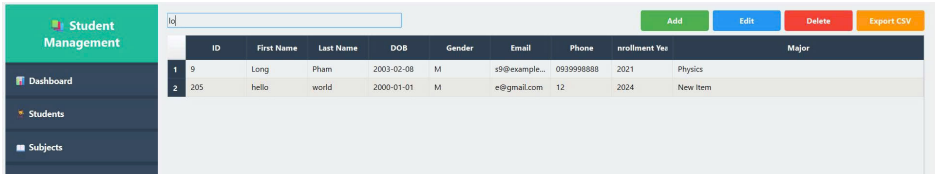

Figure 9: Search Functionality in the GUI

4.2 Required Queries

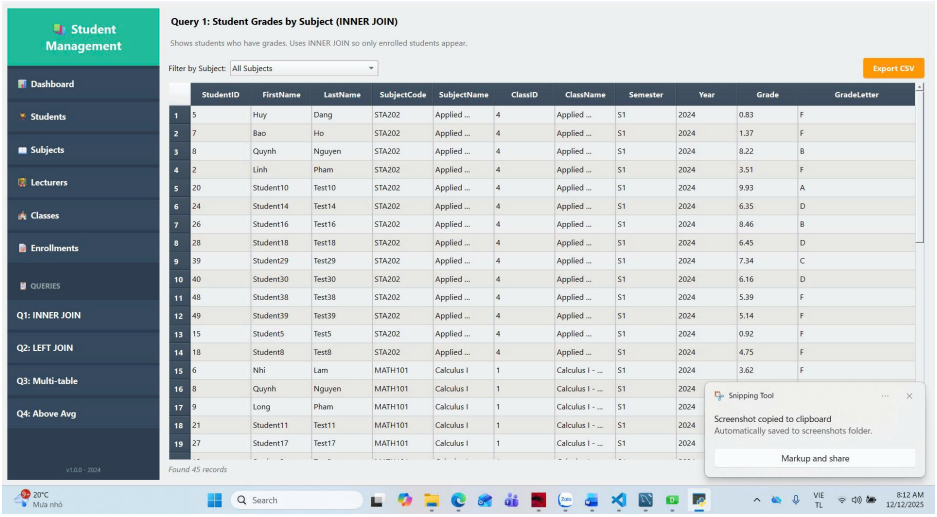

Figure 10: Query 1: Student Grades by Subject (INNER JOIN)

Explanation:

- The INNER JOIN returns only students who have at least one enrollment record.
- The query joins across `enrollments`, `students`, `classes`, and `subjects`.
- Optional filters may limit results by subject code.

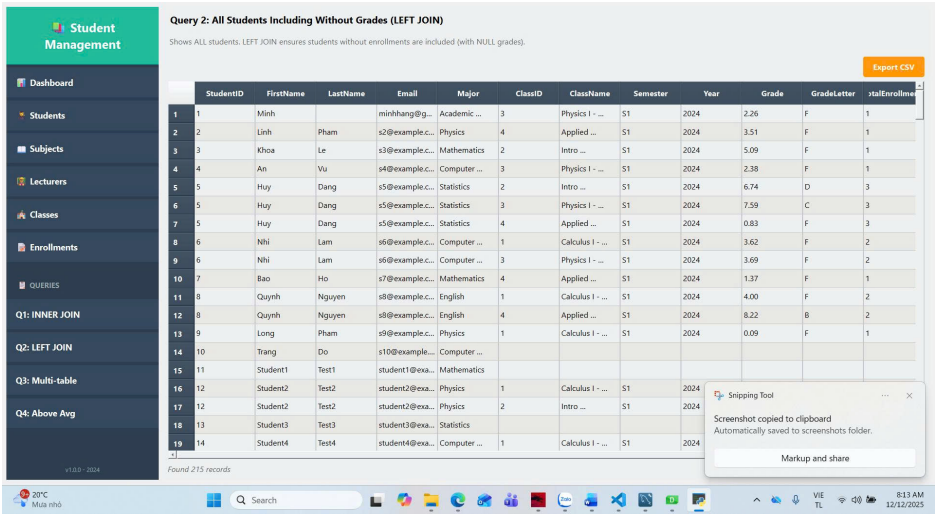

Figure 11: Query 2: All Students Including Without Grades (LEFT JOIN)

Explanation:

- The LEFT JOIN ensures all students appear in the result set.
- Students without any enrollment will show NULL values for class or grade fields.
- This query is useful for identifying students who have not registered for any subjects.

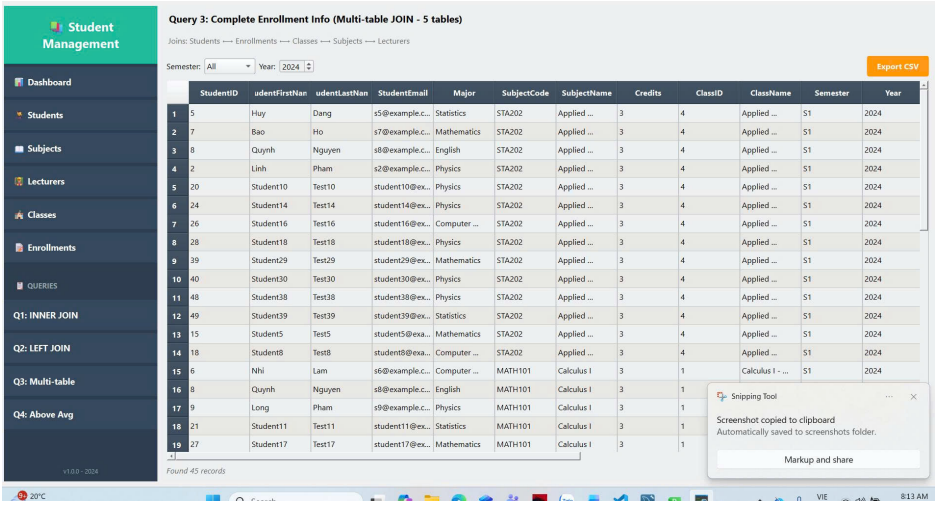

Figure 12: Query 3: Complete Enrollment Info (Multi-table JOIN, 5 Tables)

Explanation:

- The query joins five tables: `students`, `enrollments`, `classes`, `subjects`, and `lecturers`.
- It provides a full view of each enrollment, including subject and lecturer details.
- Multiple optional filters can refine the results based on subject, lecturer, or semester.

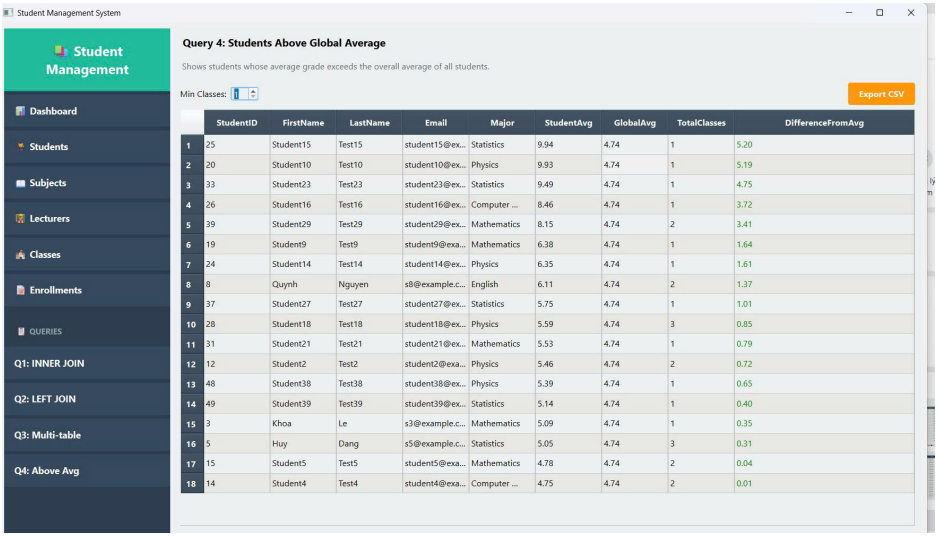

Figure 13: Query 4: Students Above Global Average

Explanation:

- The query computes the global average grade across all students and subjects.
- It also calculates each student's individual average.
- Students are filtered by comparing their average with the global mean.
- A minimum enrollment count can be applied to ensure fair comparison.

4.3 Dashboard

Figure 14: Dashboard Overview: Key Performance Indicators

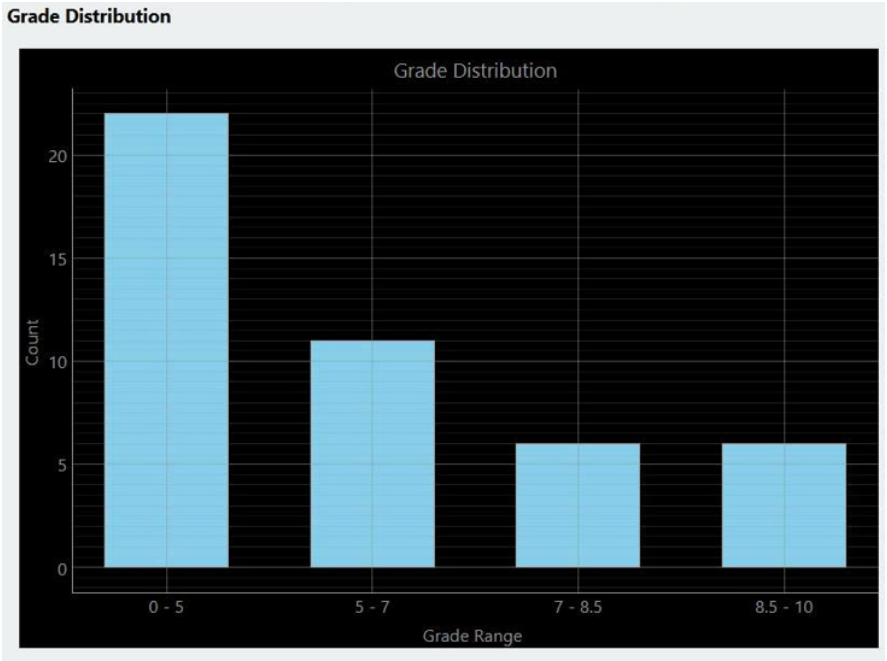

Figure 15: Grade Distribution Chart

Top 10 Students

	Rank	Name	Avg Grade	Classes
1	1	Student15 ...	9.94	1
2	2	Student10 ...	9.93	1
3	3	Student23 ...	9.49	1
4	4	Student16 ...	8.46	1
5	5	Student29 ...	8.15	2
6	6	Student9 Test9	6.38	1
7	7	Student14 ...	6.35	1
8	8	Quynh Nguyen	6.11	2
9	9	Student27 ...	5.75	1
10	10	Student18 ...	5.59	3

Figure 16: Top 10 Students by Average Grade

4.4 Testing and Error Handling

These screenshots show how the system responds when a user attempts an invalid action such as editing or deleting without selecting an item.

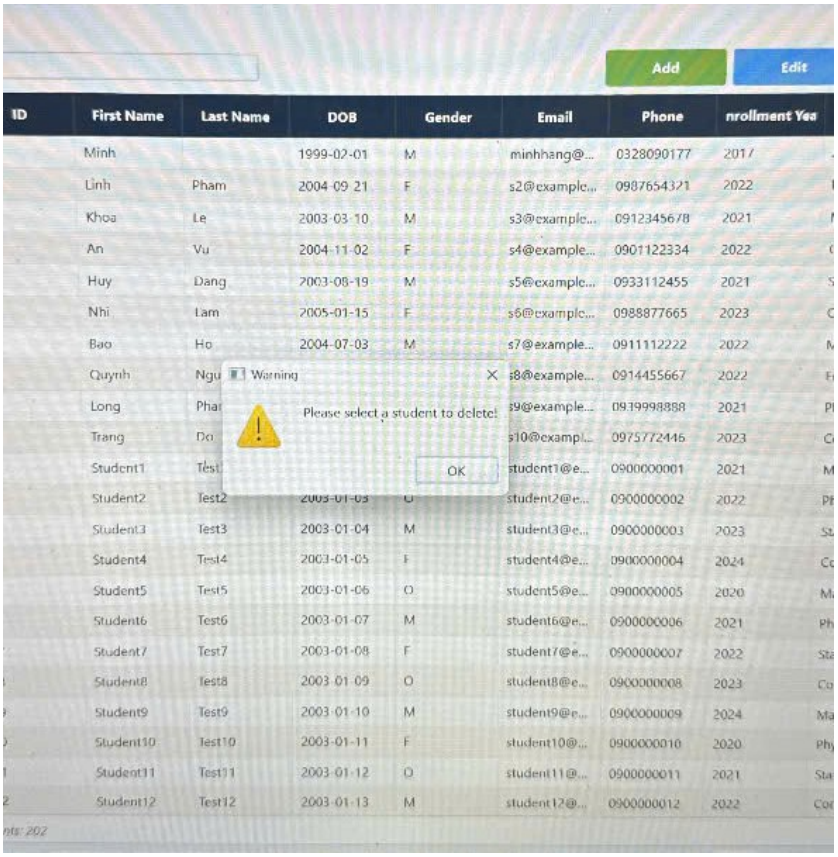

Figure 17: Delete Error: No object selected

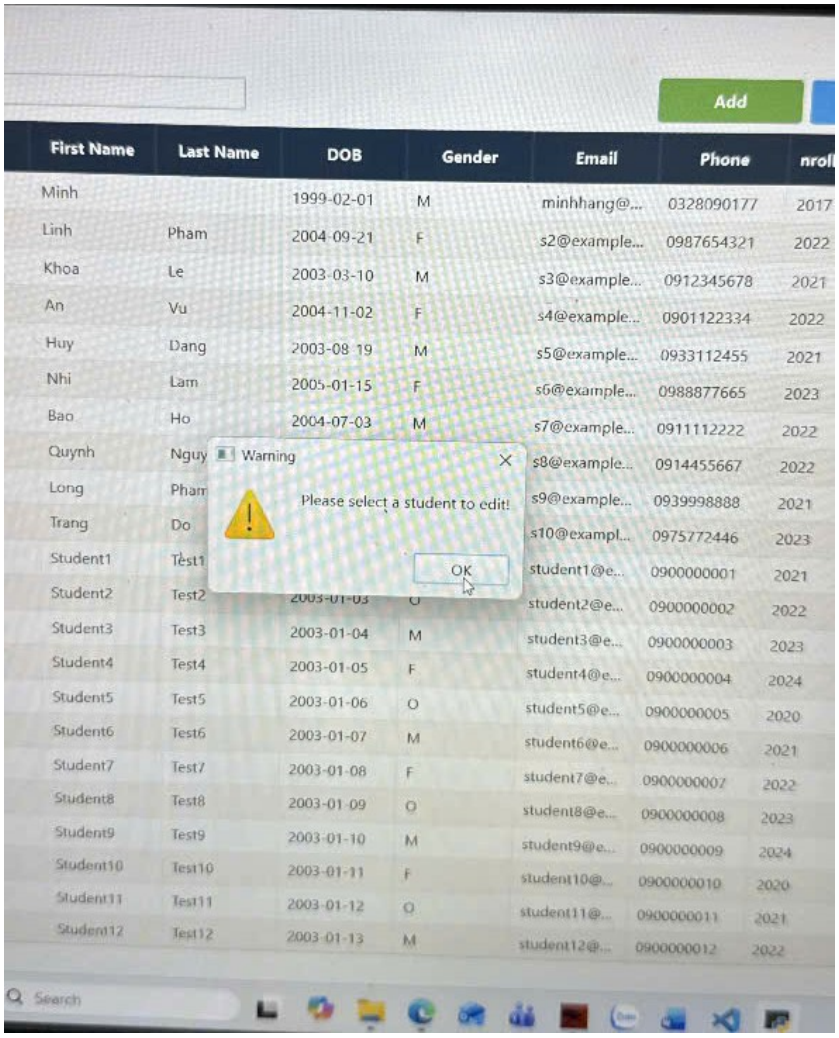

Figure 18: Edit Error: No object selected

5 Discussion

The system demonstrates a high degree of data correctness, largely due to the normalization steps applied during schema design. By decomposing the dataset into relations that satisfy Third Normal Form (3NF), the project effectively removes redundancy and eliminates anomalies such as inconsistent updates or duplicated records. The use of clearly defined primary and foreign keys further reinforces referential integrity, ensuring that operations involving students, lecturers, subjects, and enrollments behave predictably across the entire schema.

From a usability perspective, the GUI provides a practical interface for interacting with the underlying MySQL database. Its form-based structure simplifies core operations such as inserting, updating, and searching records, enabling users to work with the system without requiring direct SQL knowledge. The layout is intuitive for first-time users, and the separation of functional modules reduces confusion during CRUD operations. Although minimalistic, the interface maintains clarity and supports the educational purpose of the application.

In terms of performance, the system handles routine queries and record updates reliably under typical academic-scale workloads. The normalized schema keeps tables compact, which helps maintain responsive query times even as the dataset grows. Error handling within the GUI contributes to overall stability by preventing invalid inputs from entering the database. While the system does not yet incorporate indexing or advanced optimization techniques, the current design remains efficient for the scope of this project. Reliability is further supported by consistent database connectivity routines and structured SQL execution patterns in the backend.

Overall, the project achieves its core objectives: maintaining clean and correct data, providing an accessible user interface, and delivering dependable performance for the expected use cases of a student information management tool.

6 Conclusions

This project demonstrates a full cycle of relational database design, implementation, and evaluation. The work progressed from initial data exploration to a normalized schema, followed by the development of a functional system with clearly defined entities, relationships, and operations.

Achievements

The database was successfully normalized to higher normal forms, reducing redundancy and improving data integrity. Core modules for managing students, lecturers, subjects, classes, and enrollments were implemented with consistent CRUD operations. Constraints such as primary keys, foreign keys, and cascading behaviors were designed to maintain referential integrity across the system. The final schema supports efficient queries and provides a stable foundation for application-level features.

What Was Learned

Throughout the project, several important concepts in database engineering were reinforced. These include identifying functional dependencies, evaluating schema quality through normalization criteria, and translating conceptual models into implementable relational structures. Practical experience was also gained in designing SQL queries, handling transaction safety, and organizing a modular codebase that separates database logic from application logic.

Remaining Limitations

Despite its strengths, the system still has a few constraints. The current implementation does not include advanced indexing strategies, which limits optimization for large-scale datasets. Input validation and error handling on the application side remain minimal. In addition, the database lacks automated testing, making it harder to verify behavior under edge cases or high concurrency.

Future Improvements

Future iterations can incorporate secondary indexes and query optimization to improve performance. Expanding the system to support authentication

and role-based permissions would make it more suitable for real-world deployment. Implementing stored procedures or triggers could automate routine operations, while adding comprehensive test suites would increase reliability. Further enhancements to the GUI could also improve user experience and accessibility.

7 Team Reflection

7.1 Lessons Learned

Working as a group on the Student Information Manager gave everyone a clearer sense of how database-driven applications are built in practice. One of the strongest takeaways was learning how to organize collaboration so that each member handled a specific part of the system, from schema design and normalization to GUI layout and backend logic. This reduced overlap and helped keep development more predictable.

The project also showed how important it is to set conventions early. Agreeing on naming rules, folder structure and SQL style made the later stages smoother, especially when different modules needed to interact. Planning the database schema first, then layering the application logic on top, helped the team understand the connection between conceptual design and the final MySQL implementation.

Using GitHub throughout the process was another useful learning experience. Creating branches for features, reviewing pull requests and resolving conflicts became part of the routine. This made the team more comfortable with version control and highlighted the value of writing clear commit messages and keeping changes modular. Overall, the project strengthened both technical skills and the ability to work effectively in a small development team.

7.2 Challenges

Several challenges surfaced during development. Early in the project, inconsistencies in coding style and SQL formatting caused confusion, which meant extra time spent refactoring for consistency. The team eventually solved this by agreeing on a unified code style and rewriting a few modules to match the structure of the project.

Another issue came from dependencies between features. Modules like enrollment and class management relied on database tables that were still evolving. This led to moments where parts of the GUI or backend had to wait for the schema to stabilize. The team handled this by coordinating changes more carefully and updating documentation as the schema evolved.

GitHub also introduced some difficulties. Merge conflicts appeared when multiple members edited the same Python files or modified the schema simultaneously. Resolving these conflicts took time but reinforced good habits like smaller commits and more frequent updates.

Finally, balancing normalization theory with practical application requirements was harder than expected. Some tables worked well in higher normal forms but needed slight adjustments to function smoothly inside the GUI. Finding that balance required several iterations, but it ultimately improved the team's understanding of relational design in real applications.

References

[1] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[2] W. Kent, "A simple guide to five normal forms in relational database theory," *Communications of the ACM*, vol. 26, no. 2, pp. 120–125, 1983.

[3] P. Chen, "The entity-relationship model: Toward a unified view of data," *ACM Transactions on Database Systems*, vol. 1, no. 1, pp. 9–36, 1976.

[4] C. J. Date and H. Darwen, *Database Systems: An Implementation-Free Approach*. Addison-Wesley, 2019.

[5] R. Elmasri and S. Navathe, *Fundamentals of Database Systems*, 7th ed. Pearson, 2016.

[6] E. Freeman and E. Robson, *Head First Design Patterns*. O'Reilly Media, 2020, useful for designing GUI structure and modular applications.

[7] O. Corporation, *MySQL 8.0 Developer Guide*, 2024, available at MySQL Documentation.

[8] H. M. Pries and G. Vossen, *MySQL Cookbook: Solutions for Database Developers and Administrators*. O'Reilly Media, 2021.