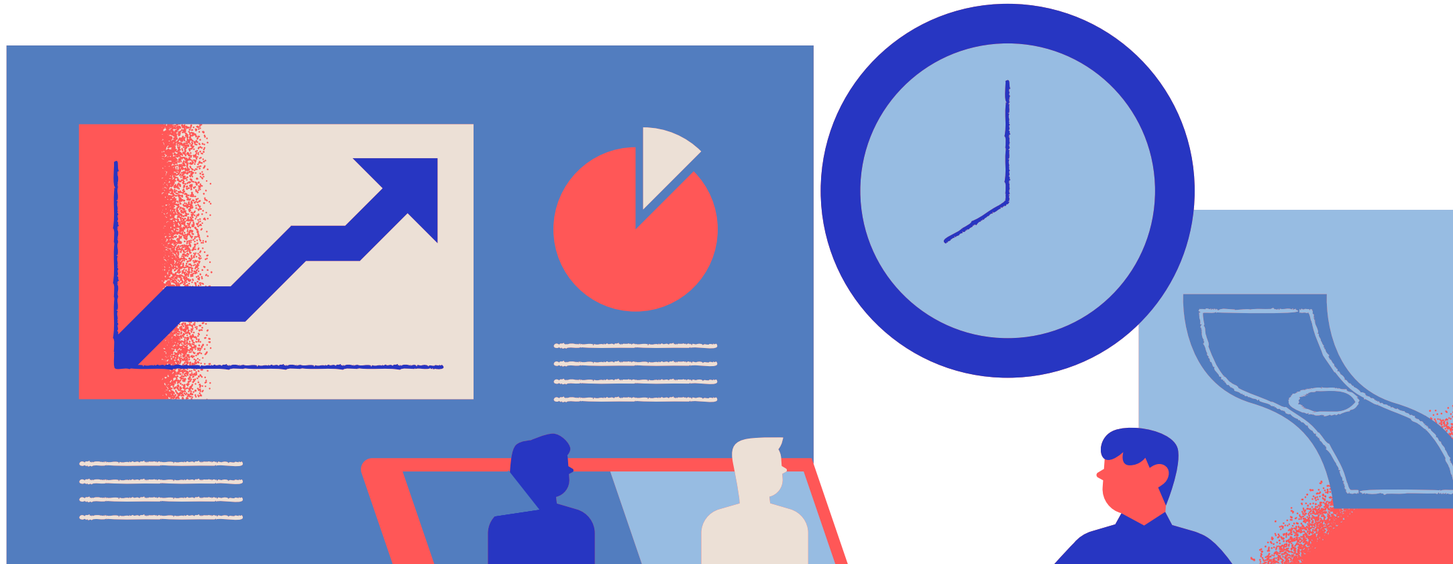
Student Information Manager

Prepared by
Nguyen Van Tue
Pham Huy Thanh
Le Duy Quyen

Group 6 - AI66 B

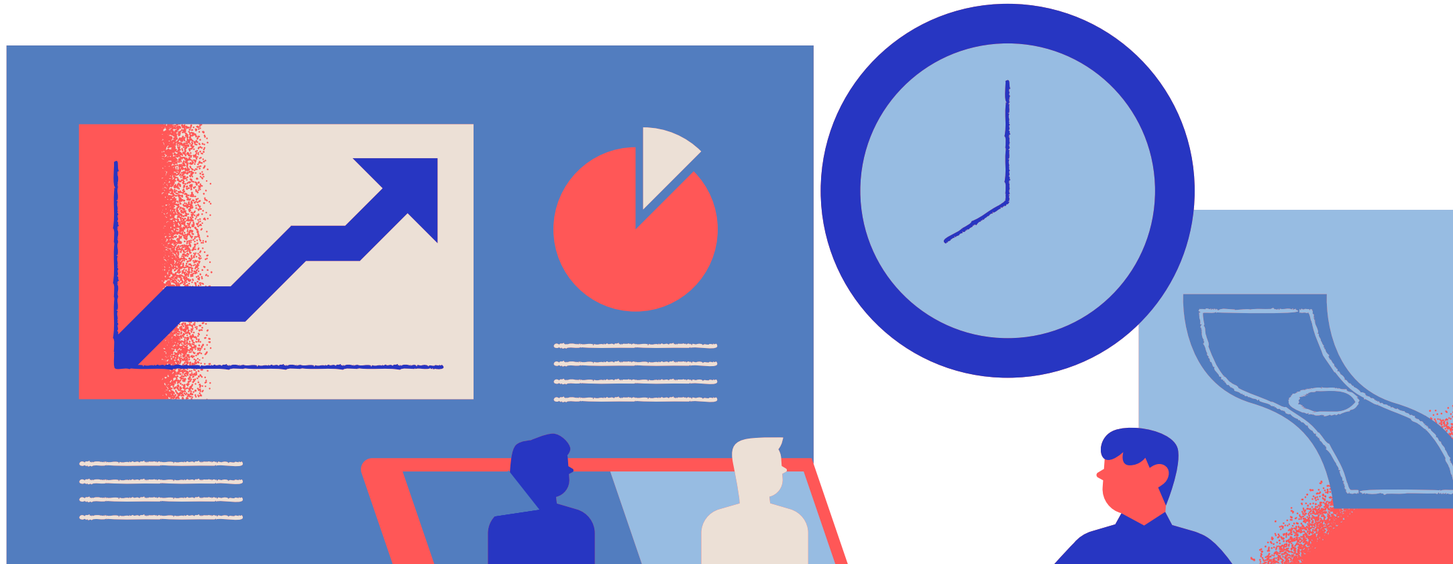
Abstract

- Designed and implemented a full-stack relational database system for academic data management.
- Transformed raw datasets through rigorous normalization (3NF) to remove redundancy and ensure referential integrity.
- Developed a complete ERD and optimized logical schema for Students, Lecturers, Subjects, Classes, and Enrollments.



Abstract

- Implemented a modular Python backend with clean architecture, parameterized SQL, and enforced business rules (e.g., prevent duplicate enrollments).
- Built a user-friendly GUI enabling intuitive CRUD operations.
- Integrated analytics: dynamic dashboards, KPIs, and SQL-driven insights for decision support.
- Achieved strong data consistency, secure operations, and improved performance across the system.



Introduction

Problem Context

- Academic data often begins in unnormalized, semi-structured formats.
- Records may contain multiple entities in one row, non-atomic attributes, or repeating groups.
- This leads to major integrity problems:
  - Redundancy
  - Insertion anomalies
  - Update anomalies
  - Deletion anomalies

Problem Context

- Such issues reduce data reliability, slow down queries, and make systems harder to scale.
- A rigorously normalized database (up to 3NF) is required to ensure:
  - Logical consistency
  - Minimal redundancy
  - Stable query performance
- This project focuses on transforming UNF → 3NF using functional dependencies and structured decomposition.

Project Scope

Category	Scope Description
Normalization	Full pipeline UNF → 1NF → 2NF → 3NF with FD derivation.
Relational Schema	PKs, FKs, cardinalities, constraints, ER → relational mapping.
Backend Layer	Python (mysql-connector), modular architecture: models, services, connection handler.
GUI Application	CRUD for students, lecturers, subjects, classes, enrollments.
Analytics	SQL queries, dashboards, KPIs, performance insights.
Evaluation	Performance, consistency, error handling, improvement recommendations.

Background

Related Work

Database Normalization

- Methodology for reducing redundancy and preventing anomalies.
- Based on E. F. Codd's relational model.
- Transforms raw UNF data into structured relations.
- Uses functional dependencies (FDs) to validate and guide decomposition.
- Ensures logical consistency and efficient query performance.

Project Scope

Normal Form	Definition
UNF	Raw, unstructured data, multi-valued or repeating fields.
1NF	Remove repeating groups; enforce atomic values.
2NF	Remove partial dependencies (for composite keys).
3NF	Remove transitive dependencies; non-key attributes depend only on PK.

Relational Database Design

- Data organized as relations (tables with tuples and attributes).
- Based on set theory and predicate logic.
- Structural constraints ensure correctness.
- Primary Keys: guarantee unique identification.
- Foreign Keys: enforce referential integrity.
- Prevents orphan records, duplication, and inconsistent updates.
- Supports data independence and efficient transactions.
- Provides a stable, reliable foundation for SQL queries and schema design.

Graphical User Interfaces

- Simplifies user interaction, no need for SQL.
- Provides visual components (forms, buttons, tables).
- Supports CRUD for key entities: Students, Lecturers, Subjects, Classes, Enrollments.
- Frameworks: Tkinter, PyQt, CustomTkinter.
- Benefits:
  - Better usability & accessibility
  - Fewer user errors
  - Shows system practicality
  - Integrates smoothly with backend logic

Methodology

Normalization

- Begins with analysis of UNF (raw, unstructured data).
- Identifies redundancy, repeating groups, multi-valued attributes.
- Detects insertion, update, and deletion anomalies.
- Establishes full set of Functional Dependencies (FDs).
- Guides systematic decomposition of relations.

Normalization Steps

UNF → 1NF

Enforce atomicity & remove repeating groups.

1NF → 2NF

Remove partial dependencies (composite keys).

2NF → 3NF

Remove transitive dependencies.

Final ERD

Consolidates all normalized relations into a unified model.

Defines entity structure, attributes, PKs, and FKs.

Illustrates cardinalities and participation constraints.

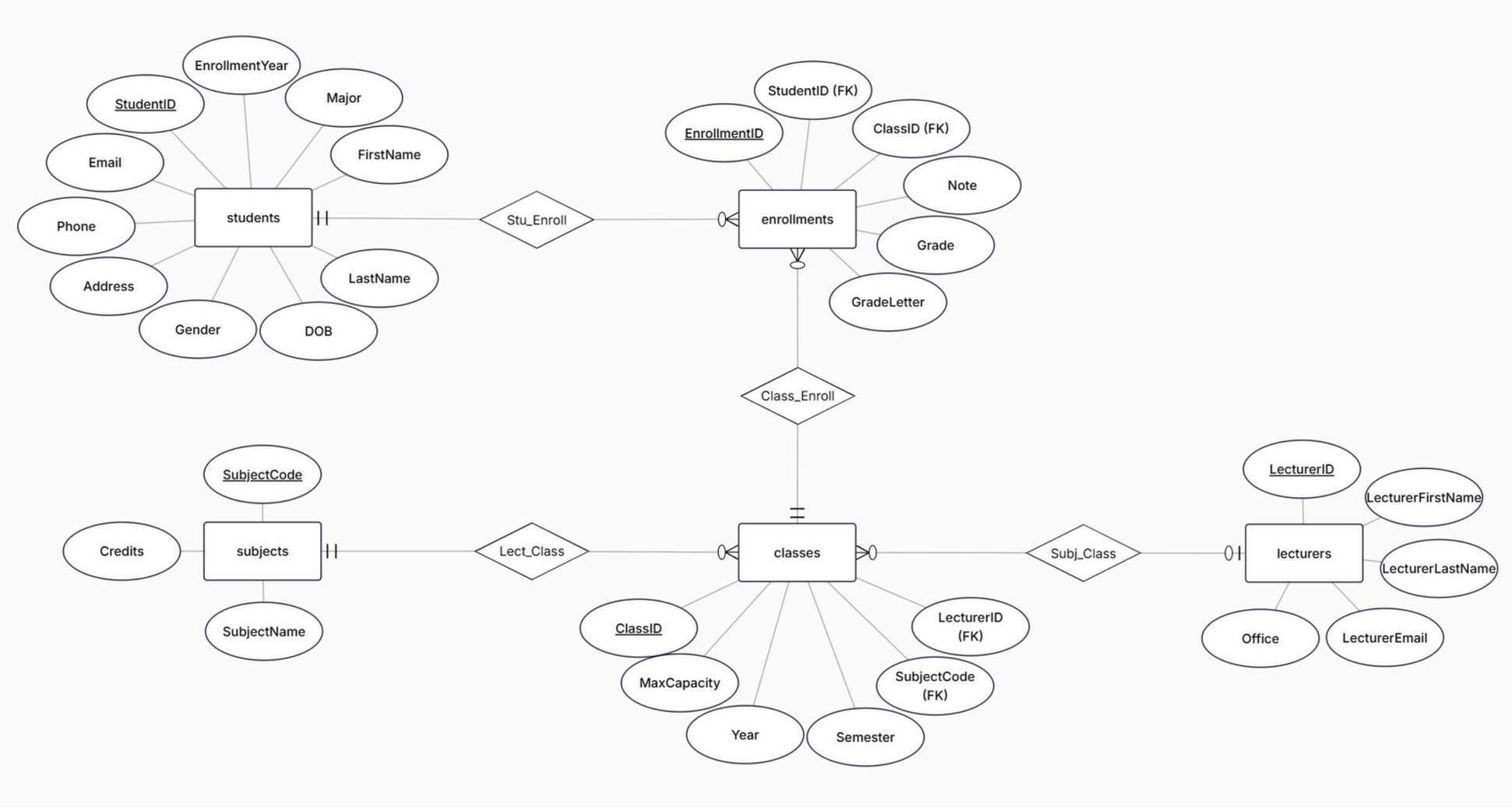
Serves as the conceptual blueprint for SQL implementation.

Accurate mapping from normalization results.

Reflects real-world academic data relationships.

Ensures referential integrity across the system.

Final ERD


EnrollmentYear
StudentID
Major
Email
FirstName
Phone
students
Address
LastName
Gender
DOB
Stu_Enroll
StudentID (FK)
EnrollmentID
ClassID (FK)
Note
enrollments
Grade
GradeLetter
Class_Enroll
SubjectCode
Credits
subjects
SubjectName
Lect_Class
classes
Subj_Class
LecturerID
LecturerFirstName
lecturers
LecturerLastName
Office
LecturerEmail
ClassID
MaxCapacity
Year
Semester
LecturerID (FK)
SubjectCode (FK)

System Architecture

Presentation Layer

- GUI components: forms, inputs, navigation
- Handles user interaction

Application Logic Layer

- Business rules & validation
- Orchestrates CRUD operations

Data Layer

- SQL execution & connection/transaction handling
- schema.sql: table creation, PK/FK, NOT NULL, UNIQUE, CHECK, indexes
- seed.sql: sample data generation (students, classes, lecturers, subjects)

Application Workflow

- GUI-based CRUD modules: Students, Lecturers, Classes, Subjects, Enrollments
- Input validation & sanitization
- Robust exception handling with clear user feedback
- Direct execution of optimized SQL queries
- Search & filtering functions
- Aggregation-based analytics
- Dashboard visualizations with KPI insights for academic monitoring

Lessons learned

Learned how normalization decisions affect the final MySQL schema.

Better understood how ERD design guides table structure and foreign keys.

Saw how GUI workflows depend on having a clean, well-organized schema.

Improved coordination while connecting Python code with SQL queries.

Became more confident using GitHub to manage database and code updates.

Future work

Add secondary indexes and optimize queries to improve overall performance.

Introduce authentication and role-based permissions for real deployment.

Use stored procedures or triggers to automate routine database tasks.

Develop a full test suite to improve stability and reliability.

Enhance the GUI to make the system easier to use and more accessible.

Thank You