Course Enrollment System

Final CS5200 DBMG Project Report

Author: Yuchen Kuang **Date:**12/10/2025

1. Introduction

This project implements a functional Course Enrollment Management System using Flask as the backend framework and PostgreSQL as the relational database.
The system supports student management, instructor management, course management, enrollment processing, soft deletion logic, capacity control, and database-level integrity enforcement through triggers and stored functions.
The primary objective of the project aligns with the CS5200 course goals:

- Designing a normalized relational schema
- Creating tables with primary keys, foreign keys, and constraints
- Implementing server-side logic with SQL, DML, views, triggers, and stored functions

Demonstrating full CRUD functionality through a working web application
The project simulates a real university environment where administrators manage students, instructors, academic departments, courses, teaching assignments, and student enrollments across semesters.

2. Requirements Overview

The project satisfies the core requirements defined in the CS5200 Final Project specification:

2.1 Required Entities

The database includes the following tables:

- **Students**
- **Instructors**
- **Departments**
- **Semesters**
- **Courses**
- **Enrollments** (associative entity for Student–Course–Semester)

Each table contains a primary key and appropriate constraints.

2.2 CRUD Requirements

The application provides GUI-based operations for:

Students

- Create, edit, view, soft delete
- Selecting courses and viewing enrollment history
- View GPA after enrolled course release grade

Instructors

- Create, edit, view, soft delete
- Automatic course deactivation when an instructor becomes inactive

Courses

- Create, edit, view, soft delete
- Prevent deletion if students are enrolled
- Force delete option with cancellation logic

Enrollments

- Add enrollment per student
- Prevent duplicate enrollment in the same course and semester

Edit grades

- Filter active vs all enrollments

2.3 Server-Side Behavior & Business Rules

This project fulfills the behavioral rule requirement through database functions and triggers:

- **Capacity enforcement**

  Trigger prevents enrollment if course capacity is full.
- **Soft delete propagation**

  When a student becomes inactive → all enrollments change to Withdrawn.

  When a course is force deleted → all active enrollments change to Course_Cancelled.
- **Grade logic**

  Grade can only be assigned or edited through a controlled interface.
- **GPA maintenance trigger** (optional but implemented)

  GPA recalculates when a student completes graded courses.

This satisfies the requirement for significant server-side logic beyond basic CRUD.

3. High-Level System Architecture

3.1 Backend

- Python Flask application
- Routes for Student / Course / Instructor / Enrollment modules
- Jinja2 templates for rendering views
- Modular design with separated concerns

3.2 Database Layer

A PostgreSQL relational schema:

```
Departments 1 --- Many Instructors
Departments 1 --- Many Students
Departments 1 --- Many Courses
Instructors 1 --- Many Courses
Students Many --- Many Courses (via Enrollments)
Courses  Many --- Many Students (via Enrollments)
Semesters 1 --- Many Enrollments
```

3.3 Key Database Features

- Primary & foreign keys
- Unique constraints
- ON DELETE CASCADE / RESTRICT rules
- Validation constraints (gpa range, credit limits, etc.)
- Stored functions implementing business logic
- Triggers enforcing capacity and delete behavior

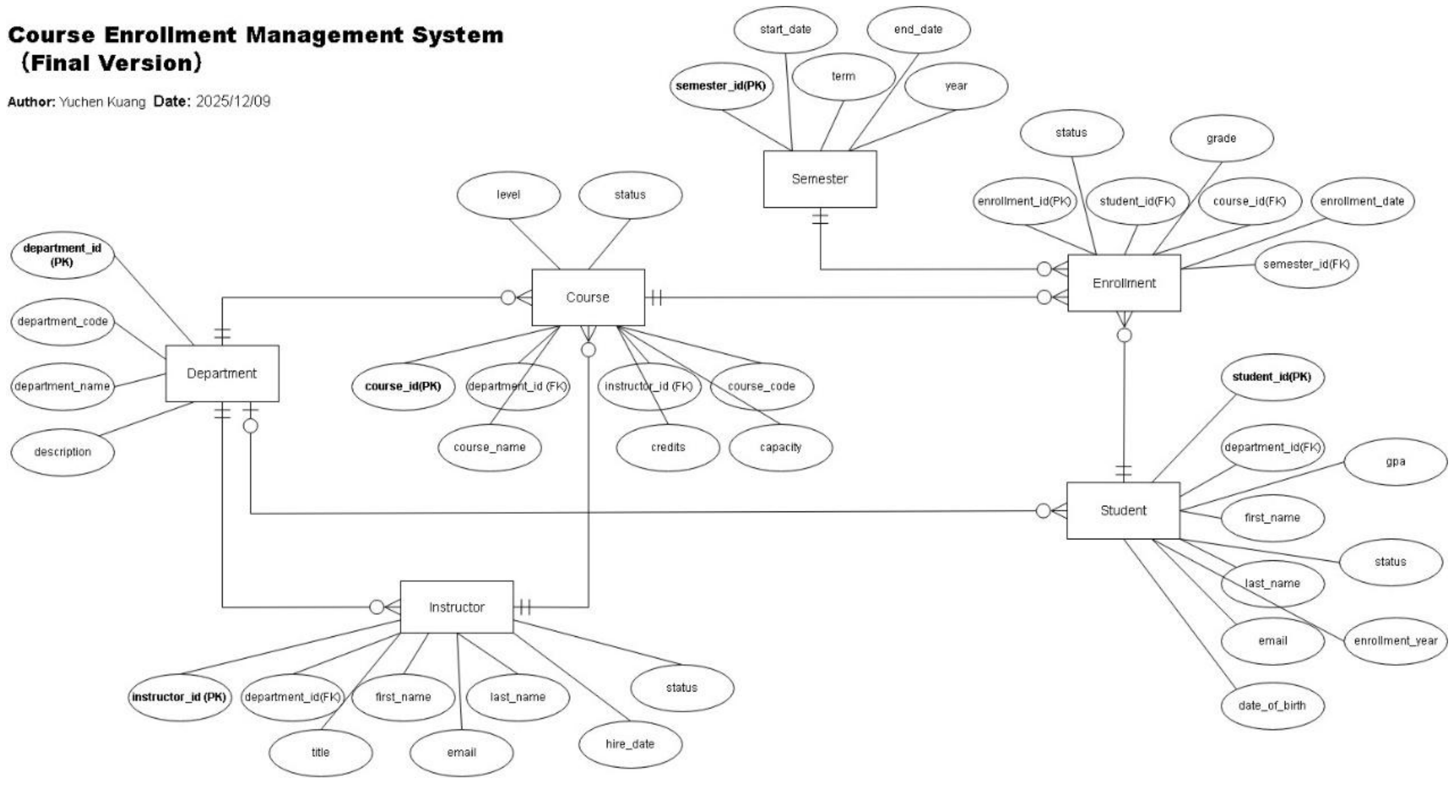
3.4 Frontend

- HTML + Bootstrap
- Navigation bar for Students / Courses / Instructors / Enrollments
- List views with filters
- Detail and edit forms
- Confirmation pages for force-deletion

4. Technical Specifications

- Language：Python + Flask
- frontend：HTML + Bootstrap
- DB：PostgreSQL
- Structure：MVC-style （routes.py – controller ＋ templates - version ＋ DB schema – data layer）
- Database features used:
    - Foreign Keys / ON DELETE CASCADE / Constraints
    - User-defined triggers
    - Multi-table joins
    - Soft delete business logic

5. Database Schema and ER Model



5.1 Overview

The database schema implements a small course enrollment management system for a university. It models five core entities — departments, instructors, students, semesters, and courses — and one associative entity, enrollments, which captures the many-to-many relationship between students and courses in a given semester.

The design enforces business rules such as unique emails, limited course capacity, valid enrollment years, and automatic GPA updates when grades are assigned.

5.2 Entities and Attributes

Departments

The departments table represents academic departments such as Computer Science or Mathematics.

Key attributes:

- department_id – primary key.
- department_code – short unique code, e.g., “CS”.
- department_name – full department name.
- description – optional descriptive text.
- created_at – timestamp when the department row was created.

Each department can have many instructors, many students, and many courses.

Instructors

The instructors table stores information about faculty members who teach courses.

Key attributes:

- instructor_id – primary key.
- department_id – foreign key to departments.department_id (NOT NULL, ON DELETE RESTRICT).
- first_name, last_name – instructor name.
- email – unique email address.
- title – job title (Professor, Lecturer, etc.).
- hire_date – date when the instructor was hired.
- status – logical status (e.g., "Active" or "Inactive") used to implement soft delete.

Each instructor belongs to exactly one department and can teach multiple courses.

Students

The students table stores student profiles.

Key attributes:

- student_id – primary key.
- department_id – optional foreign key to departments.department_id (ON DELETE SET NULL).
- first_name, last_name – student name.
- email – unique email address.
- date_of_birth – optional date of birth.
- enrollment_year – constrained by CHECK (enrollment_year >= 1900 AND enrollment_year <= EXTRACT(YEAR FROM CURRENT_DATE)) to prevent invalid future or very old years.
- gpa – numeric GPA, constrained between 0.00 and 4.00.
- status – logical status such as "Active", "Inactive", or "Graduated".

A single student can be enrolled in many courses over many semesters via the enrollments table.

Semesters

The semesters table represents academic terms (e.g., Fall 2025, Spring 2025).

Key attributes:

- semester_id – primary key.
- term – textual term label (e.g., “Fall”, “Spring”).
- year – academic year (must be ⩾ 2000).
- start_date, end_date – begin and end dates of the term, with a CHECK constraint ensuring end_date > start_date.
- A UNIQUE(term, year) constraint prevents duplicate term–year combinations.

Each semester can have many enrollments.

Courses

The courses table describes course offerings.

Key attributes:

- course_id – primary key.
- department_id – foreign key to departments.department_id, indicating which department owns the course.
- instructor_id – foreign key to instructors.instructor_id, indicating who teaches the course.
- course_code – unique course code (e.g., “CS5200”).
- course_name – descriptive course title.
- credits – number of credits, constrained to be between 1 and 10.
- description – optional course description.
- capacity – maximum number of enrolled students, must be > 0.
- level – course level (Undergraduate, Graduate, etc.).
- status – "Active" or "Inactive" for soft deletion and filtering in the application.

A course belongs to one department and one instructor, and is associated with many enrollments.

Enrollments

The enrollments table is an associative entity linking students, courses, and semesters. It represents a single student taking a specific course in a specific semester.

Key attributes:

- enrollment_id – primary key.
- student_id – foreign key to students.student_id (ON DELETE CASCADE).
- course_id – foreign key to courses.course_id (ON DELETE CASCADE).
- semester_id – foreign key to semesters.semester_id (ON DELETE CASCADE).
- enrollment_date – defaults to the current date when the row is created.
- status – logical status (e.g., "Enrolled", "Completed", "Dropped", "Course_Cancelled").
- grade – optional grade for the enrollment.
- A UNIQUE(student_id, course_id, semester_id) constraint ensures that a student cannot enroll in the same course more than once in the same semester.

This design allows the system to keep full historical enrollment records even when students, courses or instructors become inactive.

5.3 Relationships

Based on the foreign keys:

- **Department – Instructor**:
  One department to many instructors (1:N).
- **Department – Student**:
  One department to many students (1:N), with the relationship optional on the student side.
- **Department – Course**:
  One department to many courses (1:N).
- **Instructor – Course**:

One instructor to many courses (1:N).

- **Student – Enrollment – Course – Semester**:

  Many-to-many between students and courses, further refined by semester:
  - One student has many enrollments.
  - One course has many enrollments.
  - One semester has many enrollments.
  - Each enrollment binds exactly one student, one course, and one semester.

5.4 Business Rules and Server-Side Logic

The schema and database code implement several server-side business rules:

- **Data integrity constraints**
  - Unique emails for students and instructors.
  - Check constraints on enrollment year, GPA, credits, and capacity.
  - Unique combination of (student_id, course_id, semester_id) for enrollments.
- **Course capacity enforcement**
  - A trigger trg_enforce_course_capacity and function enforce_course_capacity() prevent inserting an "Enrolled" record when the number of active enrollments has reached the course capacity.
- **GPA calculation**
  - Function calculate_student_gpa() converts letter or numeric grades to grade points and computes the average.
  - Trigger trg_update_gpa automatically updates a student's gpa whenever an enrollment row is updated to status "Completed" with a grade.
- **Stored procedures for core operations**
  - enroll_student(...) encapsulates the logic to enroll a student, checking that the course exists and capacity is not exceeded.
  - drop_course(...) updates an enrollment's status to "Dropped".
  - assign_grade(...) assigns a grade, marks the enrollment "Completed", and indirectly triggers GPA recalculation.

6.User Flow Diagram

User Flow Description

The Course Enrollment System provides three main navigation modules: Students, Courses, and Instructors.

6.1 Student Management Flow

View Student List → View Student Detail

→ Edit Student

→ Soft Delete Student

↓

Enrollment records auto-updated (status = 'Dropped_Inactive')

Course capacity automatically released

Students Module

1. User opens **Student List** page.
2. User may:
   - View a student's details
   - Create a new student
   - Edit student information
   - Soft-delete a student
3. On the **Student Detail** page, the user can:
   - View all enrollment records of the student
   - Add new enrollments
   - Update or remove enrollments

Soft-deleting a student automatically updates enrollment records (status becomes *Withdrawn*) and releases course capacity.

6.2 Instructor Management Flow

View Instructor List → View Instructor Detail → Edit Instructor

→ Delete Instructor (confirmation required)

↓

Force-delete all courses taught by this instructor

↓

Each course soft-deleted → enrollments updated as Course_Cancelled

Instructors Module

1. User opens **Instructor List**, with Active/All filter.
2. From the list, the user may:
   - View instructor details
   - Create new instructor
   - Edit instructor
   - Soft-delete instructor
3. The instructor detail page displays:
   - Profile information
   - Courses taught
4. When deleting an instructor, the system requires confirmation:
   - All courses taught by the instructor are soft-deleted
   - All related enrollments are updated according to course deletion rules

6.3 Course Management Flow

```
Course List → Filter by Active / All
             → Course Detail
             → Edit Course
             → Delete Course (confirmation page)
                                  ↓
               Active enrollments → Course_Cancelled
               Course status → Inactive
```

Courses Module

1. User opens **Course List**, with filters:
   - Active Courses
   - All Courses
2. From the list, the user may:
   - View course details
   - Create a new course
   - Edit a course
   - Soft-delete a course
3. If there are enrolled students, deletion requires **confirmation**, and then:
   - All active enrollments become *Course_Cancelled*
   - Course status becomes *Inactive*

The course detail page also displays all enrolled students.

6.4 Enrollment Workflow

```
Student Detail → Enroll Student
               → Select Course & Semester
                                  ↓
               Enrollment Created (status = 'Enrolled')
               Course capacity updated

Enrollment List → Add Grade → status = Completed
               → GPA automatically recalculated (trigger)
```

Enrollment Workflow

From a student's detail page:

1. User chooses “Enroll Student”

2. Selects:
    - Course (capacity automatically checked)
    - Semester
3. Enrollment record is created with status Enrolled
4. Grades can later be added or updated
5. Enrollment List page also supports:
    - Filter: Active / All records

7. Lessons Learned

7.1 Technical Expertise Gained

Throughout this project, I strengthened my understanding of how a Flask application interacts with a relational database in a realistic, multi-entity environment. I learned how to design normalized relational schemas and how to enforce integrity using PostgreSQL constraints and triggers. Implementing business rules such as soft deletion, enrollment capacity checks, and cascading status updates required me to write meaningful SQL trigger functions rather than relying solely on application logic.

I also became much more comfortable writing multi-table joins, especially when building list and detail views that combine information from students, instructors, courses, departments, and semesters. On the Flask side, I gained practical experience organizing routes, rendering dynamic templates, handling POST/GET flows, and managing error cases cleanly. Overall, I now feel more confident building end-to-end database-driven applications where the backend and frontend work together to enforce business logic.

7.2 Insights

One of the biggest insights was understanding how complex real-world business rules become once multiple entities are involved. A simple request—such as deleting a student or removing an instructor—quickly expands into cascading implications for courses, enrollments, and system consistency. This forced me to think beyond CRUD operations and consider the lifecycle of data.

Another important insight is how strongly the backend and frontend depend on each other. For example, soft deletion only becomes meaningful when the UI clearly reflects active vs inactive records. Maintaining consistency across routes, templates, and SQL logic helped me appreciate how important it is to keep the entire stack aligned. I also learned that designing good business rules upfront saves a lot of time later.

7.3 Alternative Designs Considered

My original proposal only included Students, Courses, and Enrollments. As the project evolved, I realized the importance of additional entities such as Instructors, Departments, and Semesters. Integrating these required redesigning parts of the schema and updating UI flows so users could see department information, choose semesters for enrollments, and manage instructor assignments.

The addition of soft deletion was another major shift. Initially, the system physically deleted rows, which caused inconsistencies when courses or students disappeared while still referenced by past enrollments. Moving to a soft delete model created a much more realistic academic system and allowed historical data to remain intact.

These changes reflect how real systems evolve: new requirements emerge, and the design must adapt without breaking existing functionality.

7.4 Code Not Working / Known Limitations

While the core functionality is complete and stable, a few limitations remain:

- Instructor deletion triggers course soft deletion and enrollment status updates, but the logic assumes all courses taught by the instructor should be removed. In a real system, reassigning those courses to another instructor might be preferable.
- Some edge cases—such as re-enrolling previously dropped students, or validating overlapping schedules—are not implemented.
- There is no authentication or user role differentiation, so all operations are fully open.
- GPA calculation and grade weighting are not automated; grades exist only as simple stored fields.
- Semester creation is not exposed through the UI; only predefined semesters from the sample data can be used.

These limitations do not block the assignment requirements but would matter in a production environment.

7.5 Future Work

If I had more time, I would extend the system in several meaningful ways:

- **Authentication and Role Management**
  Implementing separate roles for admin, instructors, and students would make the application more realistic and secure.
- **Instructor and Course Re-activation**
  Currently, deleted instructors and courses remain inactive with no path to reactivate them. Adding this functionality would support better long-term data maintenance.
- **Course Scheduling and Time Conflicts**
  Real enrollment systems track meeting times and prevent students from selecting overlapping courses.
- **GPA and Academic Progress Calculation**
  With grades already stored, it would be natural to compute GPA and show students their

academic progress.

- **Reporting and Analytics**

  Useful dashboards could include enrollment statistics, capacity warnings, instructor teaching loads, and departmental summaries.

These enhancements would transform the project from a functional prototype into a more comprehensive academic management system.