**MIDTERM PROJECT**

**STUDENT MANAGEMENT SYSTEM**

**EGA RAHUL**

1. **Introduction**

This project seeks to offer a simplified portrayal of an educational institution student’s manager, mimicking the core features of a registrar’s office in learning institutions. The project is a blend of using Python language to program, MySQL database management, and HTML for the frontend representation. The simplified version of the system outlined here includes the main entities – students, courses, instructors and sections, among others. The application helps in the academic management by connecting these entities, such as academic records, course registration, and grade submission, simplifying the academic administrative works.

It outlines the SQL DDL scripts that create the shape of the database, describes optimization measures, and demonstrates the Python glue holding together the backend systems with the frontend display.

1. **SQL DDL (Data Definition Language)**

One of the important parts in SQL that is critical is the Data Definition Language (DDL) which is used for defining and structuring of databases. In this context, the SQL ‘create’ statements will be used to shape our database schema, as discussed further below. Each of these tables is constructed around different entities and their properties in accordance with the demands of our framework.

* 1. **Students Table**

CREATE TABLE students (

student\_number INT PRIMARY KEY,

name VARCHAR(255) NOT NULL,

dob DATE

);

* 1. **Courses Table**

CREATE TABLE courses (

course\_number INT PRIMARY KEY,

course\_name VARCHAR(255) NOT NULL,

credit\_hours INT,

prerequisite INT,

FOREIGN KEY (prerequisite) REFERENCES courses(course\_number)

);

* 1. **Instructors Table**

CREATE TABLE instructors (

instructor\_name VARCHAR(255) PRIMARY KEY

);

* 1. **Sections Table**

CREATE TABLE sections (

section\_identifier INT PRIMARY KEY AUTO\_INCREMENT,

course\_number INT,

instructor\_name VARCHAR(255),

semester VARCHAR(50),

year INT,

FOREIGN KEY (course\_number) REFERENCES courses(course\_number),

FOREIGN KEY (instructor\_name) REFERENCES instructors(instructor\_name)

);

* 1. **Enrollments Table**

CREATE TABLE enrollments (

enrollment\_id INT PRIMARY KEY AUTO\_INCREMENT,

student\_number INT,

section\_identifier INT,

grade VARCHAR(5),

FOREIGN KEY (student\_number) REFERENCES students(student\_number),

FOREIGN KEY (section\_identifier) REFERENCES sections(section\_identifier)

);

1. **Optimizations**

The optimization of this system ensures efficient queries while maintaining data integrity in any database systems, especially which caters for the registration process of a university. We've employed two primary mechanisms: Indexes and Foreign Keys.

**3.1 Indexes**

Indexes are used to help retrieve rows from a database table faster. These act like indices in a book and help the database system to locate the row faster than without an index. Here's how we have leveraged indexes:

**Primary Keys as Indexes**: For the table, the primary keys by default create the clustered index. The designated primary keys are on student\_number, course\_number and section\_identifier. Indexing these will make the operations faster and they are regularly searched for or accessed.

student\_number in Students table: As such, this identifier for the students ought to be indexed as it is commonly used in look-ups and joins.

course\_number in Courses table: A lot like the student number, course numbers are unique identifiers for courses and are often employed in numerous procedures.

section\_identifier in Sections table: Indexing therefore becomes all important because this is the unique identifier that identifies sections and is used to access enrollments of students.

**Auto-Increment Columns**: Attributes such as section\_identifier in the Sections table and enrolment\_id in the Enrollment table auto-increment and are unique thus allowing them to be ordered to facilitate quick retrieval of data.

**3.2 Foreign Keys**

In the database systems, referential integrity is ensured by the use of foreign keys. They make a connection between the data of two tables such that the relationship between them is preserved.

In the Courses table, prerequisite is a foreign key which is a course number. It means this relationship ensures each prerequisite listed for any course exists among the courses stored in Courses table.

The Sections table has two foreign keys:

Here, course\_number is a foreign key that refers to the course\_number in the Courses table. This makes it possible to create a section for available courses only.

instructor\_name is a foreign key referring to the primary key of the Instructors table, guaranteeing that courses are taught by registered instructors.

The Enrollments table has:

Foreign key, student\_number referencing the student\_number in the Students table to guarantee right student enrollments.

section\_identifier relates to the primary key of the Sections table and this ensures that students are enrolled only into existing sections.

1. **DML (Data Manipulation Language)**

DML or Data Manipulation Language is one of the parts of SQL which can be used to change, insert and delete data within a database. During the development of the web application, DML statements were used to load the database, update it, and even delete some data. The following DML statements were executed:

* 1. **Data Insertion**

**Inserting Data into Students Table:**

INSERT INTO students (student\_number, name, dob) VALUES

(101, 'Alice Johnson', '1999-06-15'),

(102, 'Bob Smith', '2000-03-20'),

(103, 'Charlie Brown', '2001-11-25');

**Inserting Data into Courses Table:**

INSERT INTO courses (course\_number, course\_name, credit\_hours, prerequisite) VALUES

(1, 'Introduction to Programming', 3, NULL),

(2, 'Advanced Programming', 4, 1),

(3, 'Data Structures', 4, 2);

**Inserting Data into Instructors Table:**

INSERT INTO instructors (instructor\_name) VALUES

('Dr. John Doe'),

('Prof. Jane White'),

('Dr. Emily Green');

**Inserting Data into Sections Table:**

INSERT INTO sections (course\_number, instructor\_name, semester, year) VALUES

(1, 'Dr. John Doe', 'fall', 2023),

(2, 'Prof. Jane White', 'spring', 2023),

(3, 'Dr. Emily Green', 'summer', 2023);

**Inserting Data into Enrollments Table:**

INSERT INTO enrollments (student\_number, section\_identifier, grade) VALUES

(101, 1, 'A'),

(101, 2, 'B'),

(102, 1, 'A'),

(102, 3, 'B'),

(103, 2, 'A');

1. **Python Code for Endpoints and Pages**

Below is the Python code that uses Flask framework and SQLAlchemy ORM to create routes for a web app, which is a representation of a student database. The code has arranged according to the endpoints making it easy to understand purpose of every route.

A screenshot of a computer program

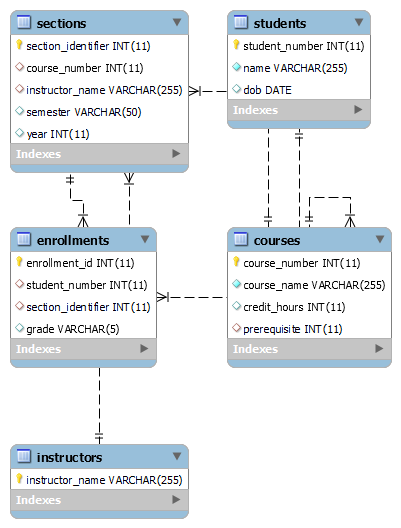
Description automatically generated

A screenshot of a computer program

Description automatically generated

The code above is the backend structure for the web applications endpoint which uses flask to render the relevant fronted pages for every endpoint.

1. **Entity Relationship (ER) Diagram Explanation**



**Entities:**

Students: This is the student’s entity with properties such as student number, name and dob.

Courses: Describes the various courses with details on their course\_number, course name, credit hours.

Instructors: Denote the instructor with one characteristic, instructor\_name.

Sections: The division of courses. There is attributes section\_identifier, course\_number, instructor\_name, semester, and year.

Enrollments: Takes the student population data for different departments. Enrollment\_id, student number, section identifier, and grade are included in this.

**Relationships:**

Students to Enrollments: A one-to-many relation, one student can have several enrollments but each is related only to the specific student.

Courses to Sections: One-to-many. A course may have several sections but each section is associated with a single course.

Courses to Courses : One-to-many recursive relationship. A course can be a prerequisite for a number of other courses; however, every course may have only one prerequisite.

Instructors to Sections: One-to-many. One instructor per section, though instructors can teach several sections at once.

Sections to Enrollments: One-to-many. Many students can be enrolled in each section, although every enrollment is related to only one section.

1. **Screenshots**

**[Homepage]**

Home page: The first page that offers a brief introduction and includes directories linking to various parts of the application.

A screen shot of a computer

Description automatically generated

**[Students List]**

View all students: Lists all students who have registered in the system and it allows to view their detailed profiles.

A screen shot of a computer

Description automatically generated

**[Student Detail]**

Student Profile: Comprehensive detail data about one person student, what about his enrolment in several sections.

A screenshot of a computer

Description automatically generated

**[Courses List]**

Course Catalog: Displays all courses offered along with course names.

A screen shot of a computer

Description automatically generated

**[Course Details]**

Course Details: This page provides an in-depth profile of a particular course and outlines its prerequisites.A screen shot of a computer

Description automatically generated

**[Instructors List]**

Instructors List: This is a page where all the instructors are listed.

A screen shot of a computer

Description automatically generated

**[Sections List]**

Sections List: Provides full details including instructors and time tables for courses listed in the particular semester.A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

**Conclusion**

I found both challenges and rewards in developing and deploying the student management system. Building an intuitive framework for tracking students, courses, instructors, and sections was the core objective. The major concern was connecting the backend SQL database with the Flask web application. This was facilitated by adopting the ORM approach through SQLAlchemy although it entailed understanding its intricacies. Also, foreign keys and an appropriate database schema were essential in upholding data consistency and connections among entities such as students and courses.

At the front-end, designing a user-friendly interface was not an easy task, as it involved trying to strike a balance between information density and ease of use. This aspect was shaped by feedback. This project was the real demonstration of the importance to start with meticulous initial design showing deeper understanding of database interactions in web application.

**YOUTUBE VIDEO:**

[**https://youtu.be/XUCHftPkrNo?si=R4Ds4oGW28l5GMse**](https://youtu.be/XUCHftPkrNo?si=R4Ds4oGW28l5GMse)