Comprehensive Academic Program Monitoring & Evaluation System

By Juan Carlos Dominguez

Program Goals and Purpose

The primary goal of our project was to design and develop a comprehensive system to evaluate degree programs within a university department. This system aimed to facilitate the collection, storage, and analysis of critical data to support ongoing program reviews. The solution needed to handle relational data, provide user-friendly interfaces, and accommodate future extensions such as new degree levels and evaluation methods.

Team Collaboration and Development Approach

This project was a collaborative effort where our team worked together on both the backend and frontend components. By leveraging our collective expertise, we ensured a seamless integration between the two layers of the application. The backend was developed using Flask and Python, which allowed efficient communication with the frontend, implemented using HTML and JavaScript.

Database Design and Implementation

We designed and implemented the database using SQL to ensure a robust and scalable structure for managing relational data. The database was engineered to store information on degrees, courses, instructors, sections, goals, and evaluations. Relationships between these entities were carefully constructed to reflect the program's requirements, such as associating courses with multiple degrees and tracking semester-specific data for course sections.

Key Features and Functionality

Our system successfully supported the following core features:

1. **Data Entry:**
    - Input of degrees, courses, instructors, sections, and goals.
    - Association of courses with program goals.
    - Entry of semester-specific course and section data.
2. **Evaluation Management:**
    - Instructors could input evaluation data for each goal associated with their courses.
    - Support for multi-degree course evaluations by duplicating data entry for multiple programs.
    - Optional inclusion of improvement suggestions for subsequent semesters.
3. **Querying Capabilities:**
    - Retrieval of degree-specific information, such as associated courses, sections, and goals.
    - Chronological listing of sections within a given time range.
    - Status tracking for evaluation completion and percentage-based performance analysis.

Achievements and Impact

Our team successfully developed a scalable and intuitive system that met all the project requirements outlined in the program description. The database and application design ensured extensibility, allowing the system to adapt to future needs. By integrating powerful querying capabilities, we empowered users to gain meaningful insights into degree evaluations and performance trends. This project demonstrated our ability to integrate backend, frontend, and database components into a cohesive solution, addressing the department's current and future program evaluation needs.

Development of the ER Model

The creation of the database schema was a multi-step process that began with the design of an **Entity-Relationship (ER) Model**. This model served as a blueprint for understanding the key entities, their attributes, and the relationships between them. The process involved collaborative brainstorming, iterative refinements, and validations to ensure a robust design.

Step 1: Identifying Entities

We started by identifying the core entities required for the project based on the functional requirements:

1. **Departments**: To represent academic departments offering degrees and courses.
2. **Degrees**: To capture details of academic programs offered by departments.
3. **Courses**: To store information about the courses offered in the curriculum.
4. **Goals**: To represent learning objectives or program-level outcomes.
5. **Instructors**: To store details about faculty members involved in teaching.
6. **Sections**: To capture semester-specific information about course offerings.
7. **Evaluations**: To track assessments of program goals through assignments.

Step 2: Defining Attributes

Each entity was analyzed to identify its attributes. For example:

- **Courses**: Attributes like `course_id`, `course_name`, `core_class`, and `dept_code`.
- **Degrees**: Attributes like `degree_id`, `degree_name`, `level`, and `dept_code`.
- **Goals**: Attributes like `goal_code` and `goal_desc`.

Special care was taken to define unique identifiers (primary keys) for each entity to ensure data integrity.

Step 3: Establishing Relationships

Once the entities were defined, we focused on the relationships between them:

- **Departments and Degrees**: A department can offer multiple degrees, but each degree belongs to one department.
- **Degrees and Goals**: Each degree is associated with multiple goals, representing program-level objectives.
- **Goals and Courses**: A goal can be addressed by multiple courses, and a course can contribute to multiple goals.
- **Instructors and Sections**: Instructors are assigned to teach sections of courses.
- **Sections and Evaluations**: Evaluations assess the achievement of goals within a specific section.

We also considered cardinality (one-to-one, one-to-many, and many-to-many relationships) to accurately capture the real-world associations.

Step 4: Refining the Model

We conducted multiple iterations of the ER Model, receiving feedback from team members and stakeholders. During this phase, we:

- Added constraints like `UNIQUE` and `CHECK` to enforce business rules.
- Simplified many-to-many relationships by introducing intermediary tables (e.g., `Goal_Courses` and `Degree_Goals`).
- Validated the model against real-world scenarios to ensure its practicality.

Step 5: Translating to Relational Schema

The finalized ER Model was converted into a relational schema. Each entity became a table, and relationships were implemented using **foreign keys**. For example:

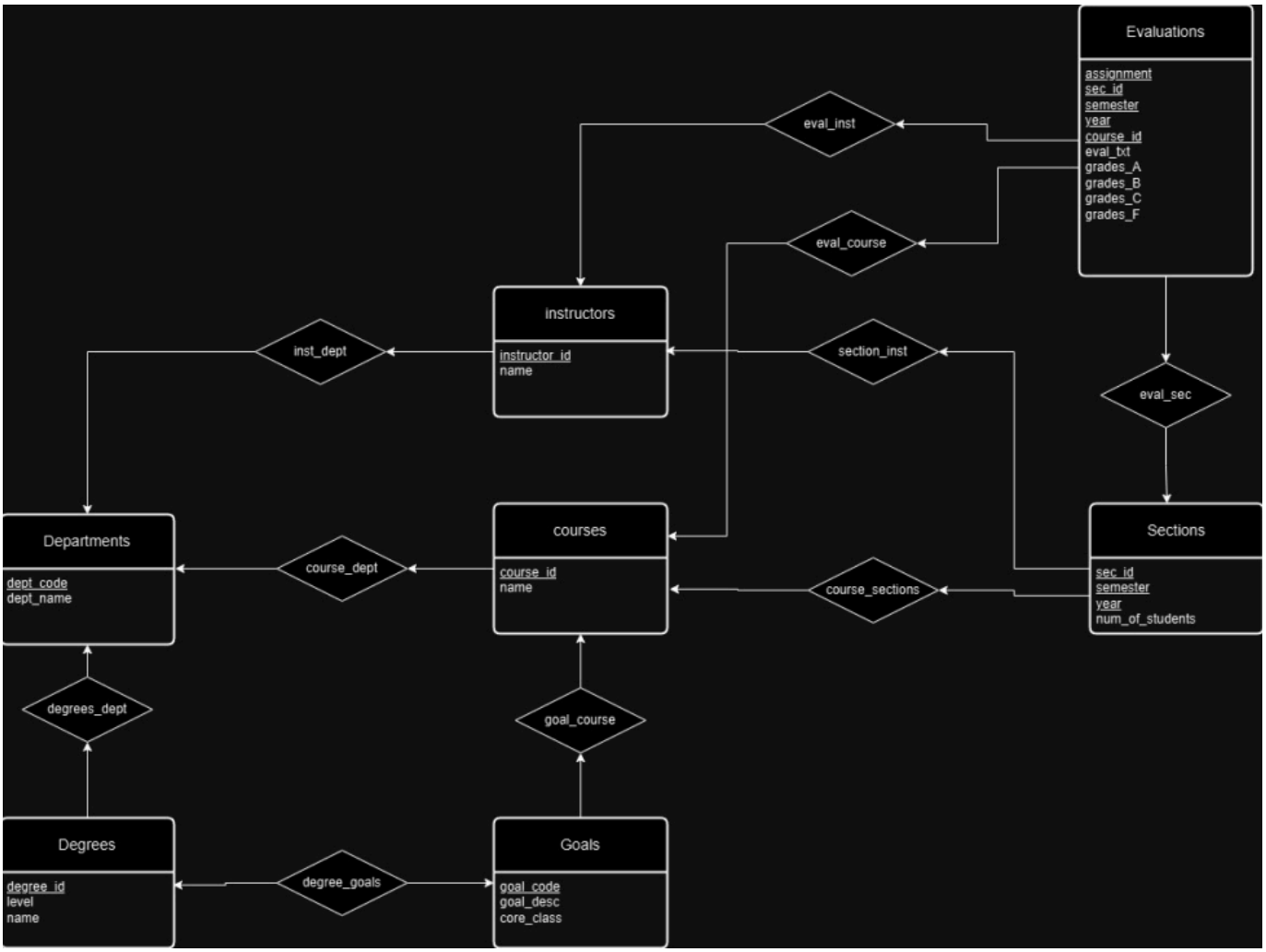
- The relationship between `Courses` and `Departments` was represented by a foreign key `dept_code` in the `Courses` table.
- The many-to-many relationship between `Goals` and `Courses` was implemented using the `Goal_Courses` table.

Step 6: Implementation and Validation

After the schema was implemented, we populated the tables with sample data and tested the relationships through SQL queries. This helped us verify the integrity of the design and ensure the schema met the project requirements.

Outcome

This process of developing the ER Model and translating it into a schema ensured that the database was both well-structured and aligned with the project's goals. The collaborative nature of the task allowed us to identify potential issues early and create a schema capable of handling complex queries and data relationships efficiently.



The following is the `University Schema` to hold our data:

```
# Dept Table -- Departments Table: Stores information about academic departments.
CREATE TABLE Departments (
        dept_code VARCHAR(2) PRIMARY KEY, -- Unique code for each department (e.g., 'CS', 'EE').
    dept_name VARCHAR(100) -- Full name of the department (e.g., 'Computer Science').
);


# Degrees Table -- Degrees Table: Stores details of academic degrees offered by departments.
CREATE TABLE Degrees (
    degree_id INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for each degree.
    level VARCHAR(50), -- Level of the degree (e.g., 'BS', 'MS').
    degree_name VARCHAR(100), -- Name of the degree (e.g., 'Computer Science').
    dept_code VARCHAR(2), -- Department offering the degree (foreign key).
    FOREIGN KEY (dept_code) REFERENCES Departments(dept_code),
    UNIQUE(degree_name, level) -- Ensures unique combinations of degree name and level.
);


# Courses Table -- Courses Table: Stores information about courses offered by departments.
CREATE TABLE Courses (
    course_id VARCHAR(8) PRIMARY KEY, -- Unique identifier for the course (e.g., 'CS1010').
    course_name VARCHAR(100) NOT NULL, -- Full name of the course.
    core_class BOOLEAN, -- Indicates if the course is a core class.
    CHECK (course_id REGEXP '^[A-Z]{2,4}[0-9]{4}$'), -- Ensures course ID follows this specific
format.
    dept_code VARCHAR(2), -- Department offering the course (foreign key).
    FOREIGN KEY (dept_code) REFERENCES Departments(dept_code)
);

# Goals Table -- Goals Table: Stores information about learning or program goals.
CREATE TABLE Goals (
    goal_code VARCHAR(4) PRIMARY KEY, -- Unique code for each goal (e.g., 'G001').
    goal_desc VARCHAR(255) -- Description of the goal.
);

#Goal_Courses -- Goal_Courses Table: Maps courses to specific goals they address.
CREATE TABLE Goal_Courses (
        goal_code VARCHAR(4), -- Goal being addressed (foreign key).
    course_id VARCHAR(8), -- Course addressing the goal (foreign key).
    FOREIGN KEY (course_id) REFERENCES Courses(course_id),
    FOREIGN KEY (goal_code) REFERENCES Goals(goal_code),
    PRIMARY KEY (goal_code, course_id) -- Ensures unique course-goal relationships.
);


# Degree_Goals Relationship -- Degree_Goals Table: Maps goals to degrees, representing program-
level objectives.
CREATE TABLE Degree_Goals (
    degree_id INT, -- Degree associated with the goal (foreign key).
    goal_code VARCHAR(4), -- Goal being addressed (foreign key).
    FOREIGN KEY (degree_id) REFERENCES Degrees(degree_id),
    FOREIGN KEY (goal_code) REFERENCES Goals(goal_code),
    PRIMARY KEY (degree_id, goal_code), -- Ensures unique degree-goal relationships.
    CONSTRAINT unique_goal_per_degree UNIQUE (degree_id, goal_code)  -- Validates uniqueness.
);
```

```
# Instructor Table -- Instructors Table: Stores information about instructors.
CREATE TABLE Instructors (
        instructor_id CHAR(8) NOT NULL, -- Unique identifier for each instructor.
    instructors_name VARCHAR(100) NOT NULL, -- Full name of the instructor.
    PRIMARY KEY (instructor_id),
    dept_code VARCHAR(2), -- Department the instructor belongs to (foreign key).
    FOREIGN KEY (dept_code) REFERENCES Departments(dept_code)
);

# Sections Table -- Sections Table: Stores information about course sections offered in different
semesters.
CREATE TABLE Sections (
    semester ENUM('Spring', 'Summer', 'Fall') NOT NULL, -- Semester of the section.
    year YEAR NOT NULL, -- Year of the section.
    sec_id CHAR(3) NOT NULL, -- Section identifier.
    course_id VARCHAR(8) NOT NULL, -- Course being offered (foreign key).
    instructor_id CHAR(8) NOT NULL, -- Instructor teaching the section (foreign key)
    num_of_students INT CHECK (num_of_students >= 0), -- Number of students in the section.
    PRIMARY KEY (semester, year, sec_id, course_id),  -- added course_id as part of primary key
due to validation issues
    FOREIGN KEY (course_id) REFERENCES Courses(course_id), -- Unique identifier for a section.
    FOREIGN KEY (instructor_id) REFERENCES Instructors(instructor_id)
);

# Evaluations Table -- Evaluations Table: Stores evaluation details for assignments mapped to
goals.
CREATE TABLE Evaluations (
    assignment VARCHAR(100), -- Name of the assignment (e.g. 'Homework', 'Quiz', 'Final Exam',
etc).
    instructor_id CHAR(8), -- Instructor associated with the evaluation (foreign key).
    sec_id CHAR(3) NOT NULL, -- Section ID where the evaluation took place.
    semester ENUM('Spring', 'Summer', 'Fall') NOT NULL, -- Semester of the evaluation.
    year YEAR NOT NULL, -- Year of the evaluation.
    course_id VARCHAR(8) NOT NULL, -- Course being evaluated (foreign key).
    goal_code VARCHAR(4), -- Goal being evaluated (foreign key)
    Eval_txt VARCHAR(255), -- Evaluation text/feedback.
    grades_A INT DEFAULT 0, -- Count of A grades assigned.
    grades_B INT DEFAULT 0, -- Count of B grades assigned.
    grades_C INT DEFAULT 0, -- Count of C grades assigned.
    grades_F INT DEFAULT 0, -- Count of F grades assigned.
    PRIMARY KEY (assignment, sec_id, semester, year, course_id, goal_code), -- Unique evaluation
entry.
    FOREIGN KEY (semester, year, sec_id, course_id) REFERENCES Sections(semester, year, sec_id,
course_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id),
    FOREIGN KEY (instructor_id) REFERENCES Instructors(instructor_id),
    FOREIGN KEY (goal_code) REFERENCES Goals(goal_code)
);
```

Database Schema Description

This schema was designed to manage and evaluate academic programs at a university, supporting robust storage and relational analysis of key data entities such as departments, degrees, courses, instructors, goals, sections, and evaluations. Each table serves a specific purpose and is interconnected to enable efficient querying and reporting.

---

1. Departments Table (`Departments`)

- **Purpose:** Stores information about academic departments offering degrees and courses.

- **Key Columns:**
  - `dept_code`: A unique 2-character identifier for each department (e.g., "CS").
  - `dept_name`: The full name of the department (e.g., "Computer Science").

---

2. Degrees Table (`Degrees`)

- **Purpose:** Contains details of academic degrees offered by departments.
- **Key Columns:**
  - `degree_id`: A unique auto-incrementing identifier for each degree.
  - `level`: Indicates the level of the degree (e.g., "BS", "MS").
  - `degree_name`: Full name of the degree (e.g., "Computer Science").
  - `dept_code`: References the department offering the degree.
- **Features:**
  - Enforces unique combinations of `degree_name` and `level`.

---

3. Courses Table (`Courses`)

- **Purpose:** Stores details about courses offered by departments.
- **Key Columns:**
  - `course_id`: A unique identifier following a specified format (e.g., "CS1010").
  - `course_name`: The full name of the course.
  - `core_class`: Boolean indicating if the course is part of the core curriculum.
  - `dept_code`: References the department offering the course.

---

4. Goals Table (`Goals`)

- **Purpose:** Maintains a list of learning or program goals.
- **Key Columns:**
  - `goal_code`: A unique 4-character code for each goal.
  - `goal_desc`: A descriptive text explaining the goal.

---

5. Goal-Courses Table (`Goal_Courses`)

- **Purpose:** Maps program goals to the courses addressing them.
- **Key Columns:**
  - `goal_code`: References a goal from the `Goals` table.
  - `course_id`: References a course from the `Courses` table.
- **Features:**
  - Ensures unique course-goal relationships.

---

6. Degree-Goals Table (`Degree_Goals`)

- **Purpose:** Links program goals to specific degrees.
- **Key Columns:**
  - `degree_id`: References a degree from the `Degrees` table.

- `goal_code`: References a goal from the `Goals` table.
- **Features:**
  - Ensures unique goal assignments for each degree.

---

7. Instructors Table (`Instructors`)

- **Purpose:** Stores information about instructors in various departments.
- **Key Columns:**
  - `instructor_id`: A unique 8-character identifier for each instructor.
  - `instructors_name`: Full name of the instructor.
  - `dept_code`: References the department the instructor belongs to.

---

8. Sections Table (`Sections`)

- **Purpose:** Contains information about course sections offered in different semesters and years.
- **Key Columns:**
  - `semester`: The semester when the section is offered (e.g., "Fall").
  - `year`: The year of the section.
  - `sec_id`: A unique identifier for the section.
  - `course_id`: References the course being offered.
  - `instructor_id`: References the instructor teaching the section.
  - `num_of_students`: The number of students enrolled in the section.
- **Features:**
  - Enforces unique section identifiers across semesters, years, and courses.

---

9. Evaluations Table (`Evaluations`)

- **Purpose:** Tracks evaluations of goals in specific course sections.
- **Key Columns:**
  - `assignment`: The name of the assignment (e.g., "Homework").
  - `instructor_id`: References the instructor conducting the evaluation.
  - `sec_id`, `semester`, `year`: Reference the specific section being evaluated.
  - `course_id`: References the course related to the evaluation.
  - `goal_code`: References the goal being evaluated.
  - `Eval_txt`: Textual feedback for the evaluation.
  - `grades_A`, `grades_B`, `grades_C`, `grades_F`: Counts of grades assigned in the evaluation.
- **Features:**
  - Ensures unique evaluations for a specific assignment in a course-section-goal combination.

Summary

This schema provides a comprehensive framework for managing academic program data, enabling detailed tracking of relationships between departments, degrees, courses, goals, instructors, and evaluations. It ensures data consistency, supports extensibility for future requirements, and facilitates efficient reporting and analysis.

Using MySQL Workbench to Implement the Database Schema

For this project, I used **MySQL Workbench** to design and implement the relational database schema. Here's the process I followed:

1. Schema Design

I started by reviewing the project requirements and identifying the key entities for the system: **Degrees**, **Courses**, **Instructors**, **Sections**, **Goals**, and **Evaluations**. After identifying these, I mapped out the relationships between them to ensure the database would be normalized, avoiding redundancy and maintaining data integrity.

2. Creating the Schema in MySQL Workbench

Once I finalized the design, I opened MySQL Workbench and used the **SQL Editor** to write the SQL commands needed to create the tables. For example, here's the script I used to create the `Degrees` table:

```
CREATE TABLE Degrees (
    degree_id INT AUTO_INCREMENT PRIMARY KEY,
    degree_name VARCHAR(255) NOT NULL,
    degree_level ENUM('Undergraduate', 'Graduate') NOT NULL,
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES Departments(dept_id)
);
```

3. Testing the Schema

After writing the SQL scripts, I executed them in MySQL Workbench to create the tables. To test the schema, I inserted sample data into the tables. For example:

```
INSERT INTO Degrees (degree_name, degree_level, department_id)
VALUES ('B.Sc. Computer Science', 'Undergraduate', 1);
```

I then used `SELECT` queries to verify that the data was stored correctly and that the relationships between tables worked as expected.

4. Refining the Schema

During testing, I made some adjustments to meet the project requirements better. For instance, I added extra columns and constraints using `ALTER TABLE` commands when necessary to support additional functionality or edge cases.

I also ensured that constraints like `NOT NULL` and `UNIQUE` were properly used where applicable. For example, I made sure that a degree name couldn't be left empty, and that department IDs in certain tables referenced valid departments to maintain data consistency.

5. Extensibility

I designed the schema with future growth in mind. For example, I used flexible data types and enums to make it easier to add new degree levels, courses, or evaluation methods in the future. The design allows the addition of new fields or tables without significant changes to the existing structure.

6. Documentation and Visual Representation

To better understand the relationships between the entities, I created an Entity-Relationship (ER) Diagram in MySQL Workbench. The ER diagram provided a visual representation of how the tables were structured and how they interacted with each other. This diagram helped me make any necessary adjustments to the schema before finalizing it.

The ER diagram also helped me ensure that all relationships (one-to-many, many-to-many, etc.) were represented in the schema and that all entities were appropriately connected.

7. **Finalizing the Schema and Database**

Once everything was set up and tested, I finalized the schema. I then proceeded with the development of the application, which uses this database schema to manage program evaluations. The schema provides a solid foundation for data manipulation and querying through SQL, and it's designed to be flexible enough to accommodate future changes.

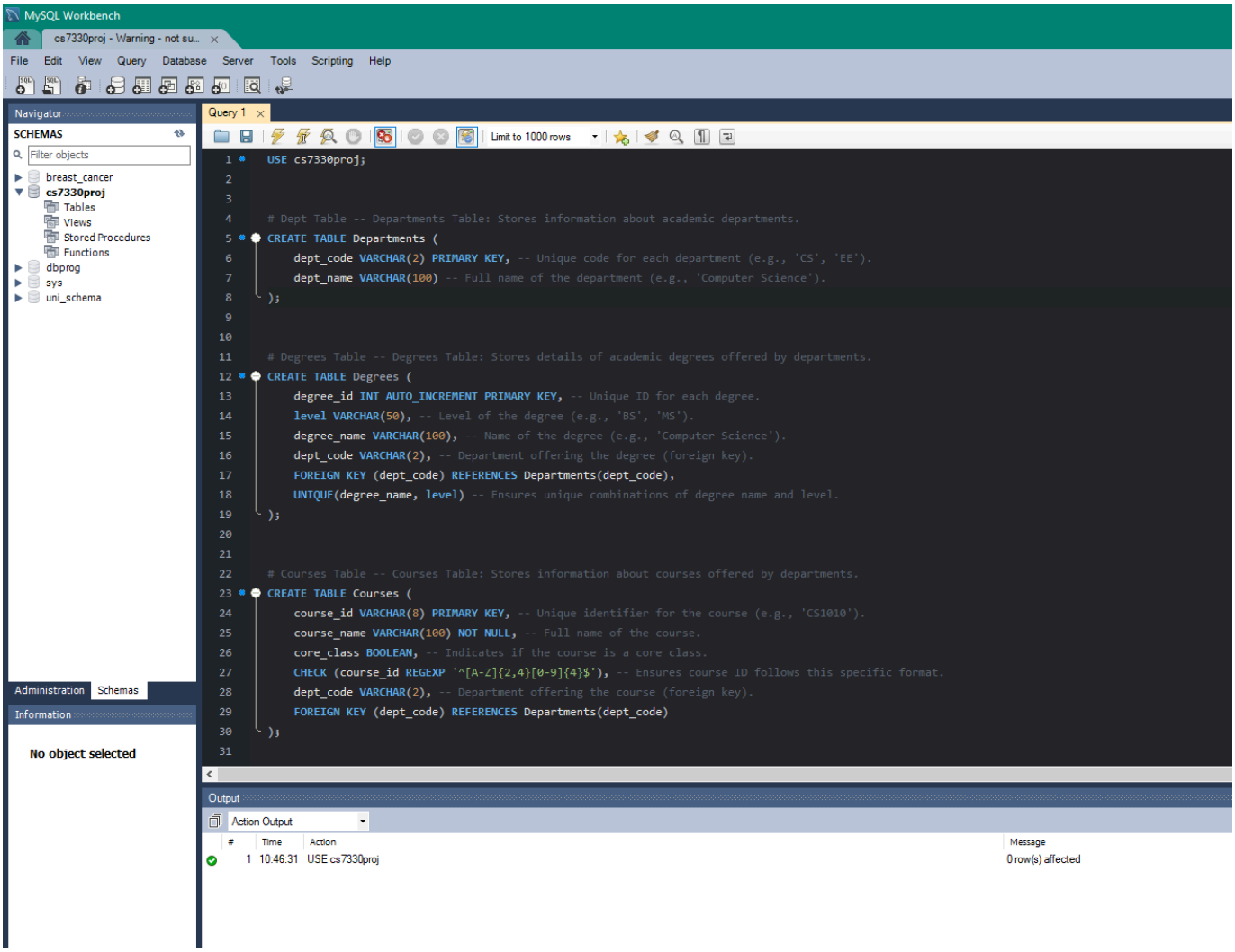
8. **Integration with the Application**

After the database schema was implemented, I integrated it into the application, allowing the frontend to interact with the database via Python and Flask. The database served as the backbone for the application, enabling functionalities like data entry, evaluation management, and querying.

The implementation also included ensuring that the database supported fast and efficient queries, especially when it came to pulling data for reports and evaluations. The schema allowed for the seamless linking of degrees, courses, instructors, and evaluation data.

9. **Conclusion**

This process of working through the schema design, testing, and finalizing in MySQL Workbench allowed me to create a solid, scalable, and extensible database system. The use of foreign keys, constraints, and normalization ensured the data integrity of the system, while the visual ER diagram made the relationships between entities clear and easy to manage. This schema forms the foundation of the application, providing a robust backend that supports data management and querying for program evaluations.

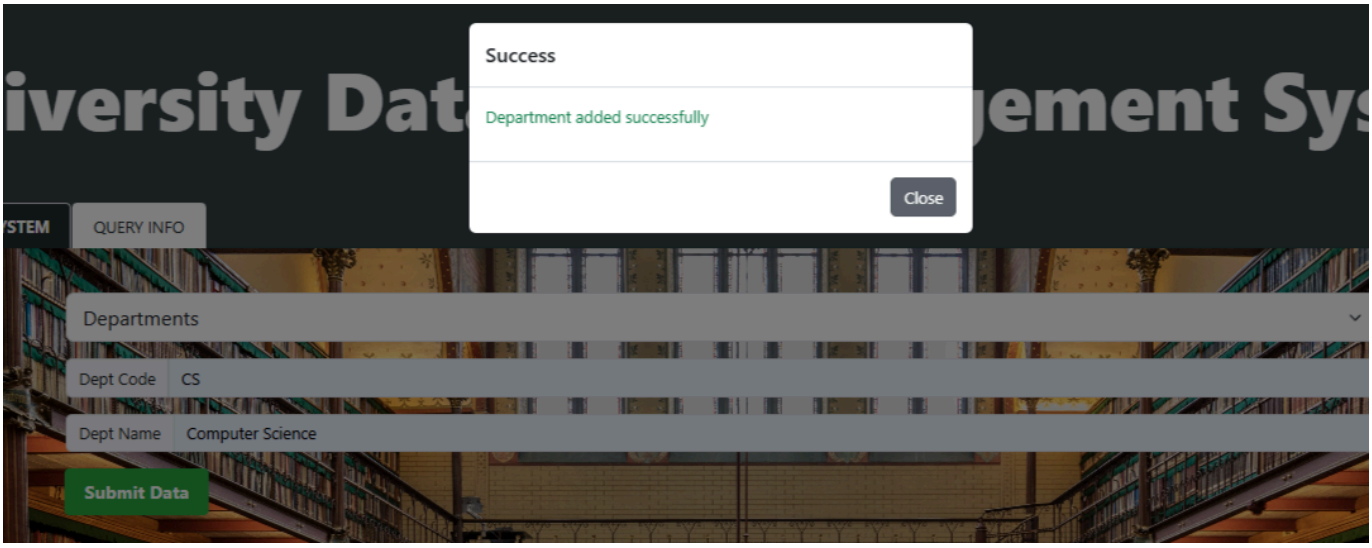


Adding Data to the System

After creating the database and tables, the next step is to add data to the system. The schema and code include constraints for each input to ensure data integrity and consistency.

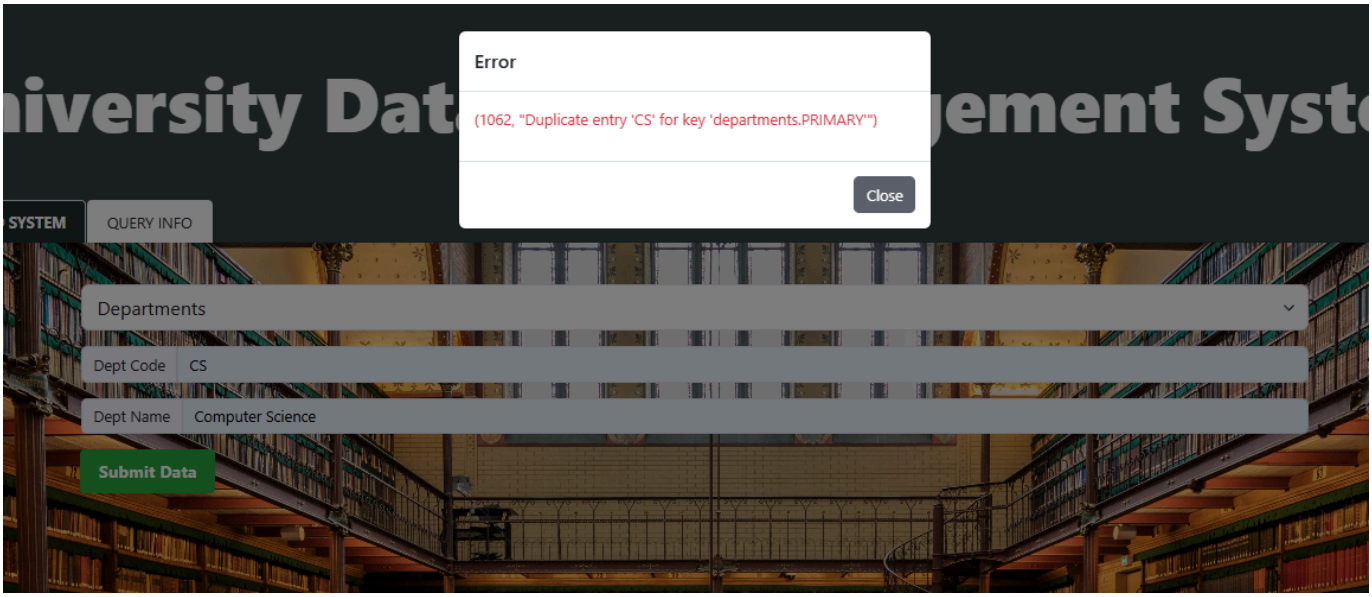
Adding Department Data

The process begins by adding the **Department Code** and **Department Name** to the `Departments` table. Following the constraints defined in the schema, the data was successfully inserted into the table.



Preventing Duplicate Entries

If an attempt is made to enter duplicate data into the table, the system raises a **Duplicate Entry Error**, demonstrating that the schema enforces data integrity and consistency effectively.



Verifying Data Entry in MySQL Workbench

To confirm that the database accepted the input, we can query the data in **MySQL Workbench**:

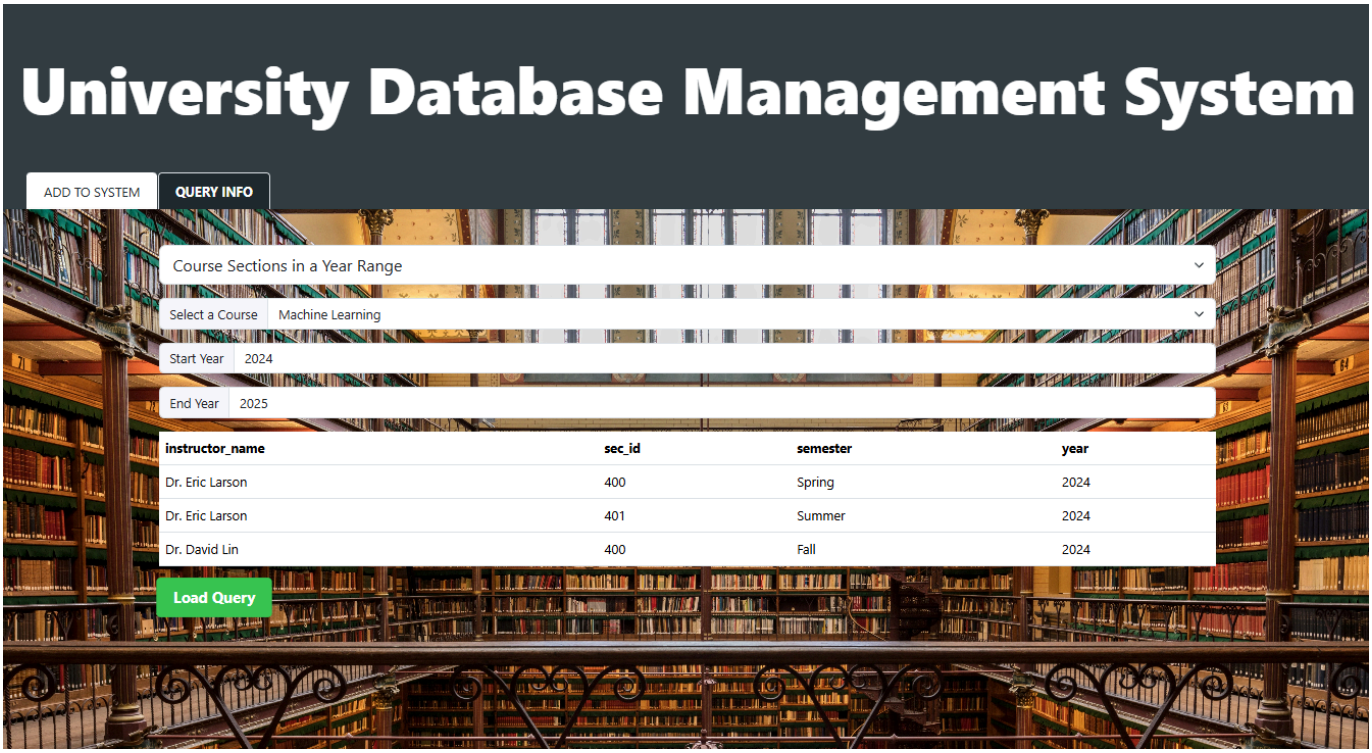
```
102
103 *   SELECT * FROM departments;
104
105
```

Result Grid | Filter Rows:

dept_code	dept_name
CS	Computer Science
NULL	NULL

Inputting and Querying Data

After entering the required data, we can proceed to query it directly from within the system:



Verifying Data in MySQL Workbench

We can also verify the same data using **MySQL Workbench**:

```
105 *   SELECT instructor_id, sec_id, semester, year
106     FROM courses c, sections s
107     WHERE c.course_id = s.course_id;
108
```

Result Grid | Filter Rows: | Export: | Wrap Cell Conter

instructor_id	sec_id	semester	year
111111	400	Spring	2024
111111	401	Summer	2024
222222	400	Fall	2024

Project Summary

This project successfully accomplished its goal of creating a robust, scalable, and extensible database system for evaluating university degree programs. By carefully designing and implementing the database schema in **MySQL Workbench**, the project ensured data integrity, consistency, and efficiency throughout the system. Here's an overview of how these objectives were achieved:

Database Design and Implementation

1. **Schema Design**:
    - The database schema was normalized to avoid redundancy and maintain referential integrity.
    - Key entities such as **Degrees**, **Courses**, **Instructors**, **Sections**, **Goals**, and **Evaluations** were carefully designed, with relationships and constraints (e.g., `FOREIGN KEY`, `NOT NULL`, and `UNIQUE`) ensuring accurate and consistent data storage.
2. **Data Integrity and Consistency**:
    - Constraints in the schema prevented invalid data entries (e.g., duplicate entries, null values in critical fields).
    - Testing with sample data validated the integrity of relationships and ensured the database operated as expected.
    - The use of `ENUM` data types and flexible design choices allowed for easy future expansion without disrupting existing data or workflows.
3. **Performance Optimization**:
    - Indexing and careful query structuring ensured efficient data retrieval, particularly for complex reports and queries involving multiple relationships.

Backend and Frontend Integration

The integration of **Python** and **Flask** with **JavaScript** enabled seamless interaction between the application frontend and the SQL database. Key highlights include:

1. **Backend Functionality**:
    - Flask provided RESTful APIs to handle database operations like data entry, updates, and querying.
    - The backend ensured proper handling of SQL queries to prevent injection attacks and other vulnerabilities.
2. **Frontend Interaction**:
    - JavaScript was used for dynamic, responsive interfaces, allowing users to input and query data efficiently.
    - Asynchronous JavaScript (AJAX) enabled real-time interactions, minimizing reloads and improving user experience.
3. **Performance Monitoring**:
    - Continuous testing ensured the SQL database responded efficiently to requests, even with large datasets.
    - The use of Flask as middleware streamlined communication between the frontend and the database, maintaining speed and reliability.

Maintaining Data Integrity and Consistency

- **Validation at Multiple Levels**: Data validation occurred both at the frontend (e.g., form input checks via JavaScript) and backend (e.g., schema constraints, input sanitization in Flask).
- **Error Handling**: Comprehensive error messages guided users when invalid inputs were detected, preventing data corruption.
- **Auditing and Testing**: Regular schema tests and real-time querying ensured that the database maintained its integrity even as new features or updates were added.

Conclusion

This project provides a solid foundation for managing program evaluations, ensuring that the system is reliable, extensible, and capable of handling complex relationships and queries. By integrating a well-structured SQL database with a Python-Flask backend and a dynamic JavaScript-powered frontend, the application delivers a seamless and efficient user experience while maintaining high standards of data integrity and consistency.