



IE 401

Database Design and Implementation Project

**ONLINE COURSE RATING MANAGEMENT
DATABASE SYSTEM PROJECT**

**Elif İlayda Güntürk, Enes Orman, Ezgi Su Bilal, Hülya İkranur Diltemiz, Turgut Berk
Karahasanoğlu**

ACADEMIC HONESTY PLEDGE

In keeping with MEF University Student Code of Conduct, I pledge that this work is my own and that I have not received inappropriate assistance in its preparation. I further declare that all resources are explicitly cited.






| <u>NAME</u> | <u>DATE</u> | <u>SIGNATURE</u> |
|---------------------------|-------------|---|
| Ezgi Su Bilal | 3/12/2024 |  |
| Elif İlayda Güntürk | 3/12/2024 |  |
| Hülya İkranur Diltemiz | 3/12/2024 |  |
| Enes Orman | 3/12/2024 |  |
| Turgut Berk Karahasanoğlu | 3/12/2024 |  |

TABLE OF CONTENTS

| | |
|---|----|
| TABLE OF CONTENTS | i |
| 1. INTRODUCTION | 1 |
| 1.1. Project Topic | 1 |
| 1.2. Assumptions and Constraints | 3 |
| 1.3. Ethical Issues | 3 |
| 1.4. Business Rules | 4 |
| 1.5. Use Cases | 4 |
| 1.6. CRUD Operations for the Online Course Rating System | 5 |
| 2. ER MODELING | 6 |
| 2.1. Entities and Attributes | 6 |
| 2.2. ER Diagram | 8 |
| 3. Implementation | 9 |
| 3.1. Execution Procedure and SQL Scripts | 9 |
| 3.2. Alignment of Design with Requirements | 16 |
| 4. NUMERICAL EXAMPLE | 19 |
| 5. CONCLUSION | 22 |
| 5.1. Broad Impact | 22 |
| 5.2. Life-Long Learning | 23 |
| 5.3. Professional and Ethical Responsibilities of Engineers | 23 |
| 5.4. Contemporary Issues/Future of Industry | 24 |
| REFERENCES | 25 |

1. INTRODUCTION

In today's rapidly evolving digital environment, data management and automation have become key elements for the success of businesses and organizations. When we think about educational systems, especially institutions that offer courses and employ instructors, managing user data and ensuring interactions between various titles such as students, instructors, and courses is quite important. We thought it would be important to implement a database system to simplify and automate managing users and instructors.

This project aims to design a system that automatically generates unique identifiers for users, instructors, and courses using triggers to maintain data integrity and consistency. The goal is to automate the assignment of IDs and ensure proper linking of entities such as users, instructors, and courses across different database tables.

The system will handle key entities—users, instructors, courses, and ratings—while using triggers to auto-generate IDs for new entries and updates, based on user roles. This will improve efficiency, reduce errors, and maintain consistency across the platform.

The project focuses on automating data management and shows operations and user experience in an educational context.

1.1. Project Topic

This project focuses on developing a robust database system for an online course rating platform. The system allows users to rate and review courses, helping prospective students make informed decisions. The platform also enables instructors to receive feedback and improve their courses. The database supports core functionalities such as managing user accounts, courses, reviews, and ratings. The system's goals and scope for the online course rating platform are as follows:

Goals:

- 1. Enhanced Decision-Making for Students:** Provide a centralized platform where students can rate and review courses, enabling prospective learners to make informed enrollment decisions.
- 2. Constructive Feedback for Instructors:** Allow instructors to gather insights from student feedback to improve course content and teaching methods.
- 3. Efficient Data Management:** Enable seamless management of user accounts, courses, ratings, and reviews through a well-structured database.
- 4. Transparency and Trust:** Maintain the integrity of reviews by ensuring they are unaltered and linked to verified course enrollments.
- 5. Scalable and Secure Design:** Develop a scalable platform capable of handling a growing number of users and courses while ensuring data security.

Scope:

- 1. User Roles:** The system will manage two types of users, students and instructors, each with distinct functionalities and access levels.
- 2. Course Management:** Instructors can create and manage multiple courses, and students can enroll in these courses to participate and provide feedback.
- 3. Ratings and Reviews:** Students can rate courses on a scale of 1 to 5 stars and optionally add comments. Ratings below 3 stars require mandatory comments.
- 4. Data Relationships:** The database will establish clear relationships between users, courses, enrollments, and reviews to ensure consistency and data integrity.
- 5. Use Cases:** The system will support key operations such as viewing course feedback, retrieving instructor ratings, and listing enrolled students.
- 6. Scalability:** The platform will support the addition of new courses, users, and reviews, with provisions for future integration with AI-based recommendations.

1.2. Assumptions and Constraints

The system functions on a number of fundamental presumptions. The ratings are authentic because the users who are giving feedback are confirmed students or alumni. Ratings are presumed to be impartial and truthful, representing actual user experiences. It is anticipated that institutions will embrace and apply the system's findings for ongoing development. In order to preserve uniformity and integrity throughout all processes, the system also automatically creates unique identities for users, teachers, and courses.

One of the main limitations is the early emphasis on university-level courses with a particular user base, which restricts scalability. Because users can only review a course once, feedback cannot be manipulated. All reviews must adhere to the platform's rules, and any offensive remarks must be reported or deleted. Until more improvements are made, the current system architecture restricts scalability even though it is built to manage growing traffic over time.

1.3. Ethical Issues

There are important ethical issues with the development of the online course rating system. Maintaining user privacy and anonymity is essential for building trust and allowing for objective criticism. Strong encryption and access control procedures protect sensitive data, such as credentials and reviews. Through moderation procedures, the system guards against inaccurate or disparaging evaluations to maintain impartiality. By giving customers explicit instructions on how to submit reviews and outlining how comments will be used to enhance instructor evaluations and courses, transparency is given top priority.

The platform's design includes inclusivity, making it accessible to students with disabilities through features like adaptive interfaces and screen reader compatibility. Additionally, anonymizing comments and concentrating just on the caliber of teachers and courses addresses bias mitigation. Additionally, users are reminded of their responsibility to offer honest and beneficial feedback in order to preserve the platform's fairness and integrity.

1.4. Business Rules

- Each course is uniquely identified by a Course ID and must belong to one instructor.
- Users can only rate and review a course if they are enrolled.
- An instructor can be assigned to teach multiple courses.
- Courses can only be created by users with the role of "Instructor."
- Each user has a unique account with distinct roles: student or instructor.
- Comments on reviews can be null.
- Every user can take more than one course at the same time.

1.5. Use Cases

List All Courses Offered by an Instructor: Retrieve the list of courses that a specific instructor is teaching. This allows instructors to manage their portfolios or users to explore courses offered by a particular instructor.

Calculate Average Rating for a Course: Compute the average rating for a course based on student feedback. This helps potential students gauge the course's quality.

Find Students Enrolled in a Course: Display all students who are enrolled in a specific course. This allows instructors to understand their audience or verify enrollments.

Fetch Courses Without Reviews: Identify courses that have not received any reviews to encourage students to provide feedback and improve engagement.

Access Reviews with Low Ratings: Retrieve all reviews with ratings below 3 stars. This enables instructors to identify areas for improvement based on detailed feedback.

1.6. CRUD Operations for the Online Course Rating System

Create: New users and courses can be added to the database. For example, instructors can be registered, and their courses can be defined with specific attributes such as title and description.

Read: Existing data in the database can be retrieved. For instance, all reviews for a particular course can be viewed, or a list of students enrolled in a course can be obtained.

Update: Modifications can be made to existing data. For example, a user's email address can be updated, or a comment on a review can be edited to reflect more accurate feedback.

Delete: Records can be removed from the database when no longer needed. For example, users who withdraw from the platform or enroll for completed courses can be deleted.

2. ER MODELING

2.1 Entities and Attributes

1. User_Info

- **Purpose:** Represents users in the system, including both instructors and students.
- **Attributes:**
 - **user_id (PK):** A unique identifier for each user, automatically generated for every new user.
 - **name_user:** The user's full name.
 - **email:** The email address of the user, used for login and communication.
 - **password_user:** A secure password for authentication.
 - **role_user:** Indicates whether the user is an "Instructor" (*I*) or a "Student" (*S*).

2. Instructor

- **Purpose:** Represents additional details for users with the "Instructor" role.
- **Attributes:**
 - **instructor_id (PK):** A unique identifier for each instructor, auto-generated when a user's role is "I."
 - **user_id (FK):** A foreign key linking to the *user_id* in the *User_Info* table to associate an instructor with their user profile.
 - **Department:** Specifies the department the instructor belongs to.

3. Course

- **Purpose:** Represents the courses offered in the system.
- **Attributes:**
 - **course_id (PK):** A unique identifier for each course, automatically generated.
 - **name_course:** The name of the course.
 - **department:** The department to which the course belongs.
 - **instructor_id (FK):** A foreign key linking to the *instructor_id* in the *Instructor* table to identify the instructor responsible for the course.

4. Assigned_Course

- **Purpose:** Tracks the assignment of courses to users (students or instructors).
- **Attributes:**
 - **assigned_id (PK):** A unique identifier for each course assignment.
 - **user_id (FK):** A foreign key linking to the *user_id* in the *User_Info* table to identify the assigned user.
 - **course_id (FK):** A foreign key linking to the *course_id* in the *Course* table to identify the assigned course.

5. Rating

- **Purpose:** Stores user feedback and ratings for courses.
- **Attributes:**
 - **rating_id (PK):** A unique identifier for each rating entry.
 - **user_id (FK):** A foreign key linking to the *user_id* in the *User_Info* table to identify the user providing the rating.
 - **course_id (FK):** A foreign key linking to the *course_id* in the *Course* table to associate the rating with a specific course.
 - **instructor_id (FK):** A foreign key linking to the *instructor_id* in the *Instructor* table to associate the rating with a specific instructor.
 - **rating_star:** A numerical rating (1 to 5 stars) given by the user.
 - **rating_comment:** Additional feedback or comments provided by the user about the course.

2.2. ER Diagram

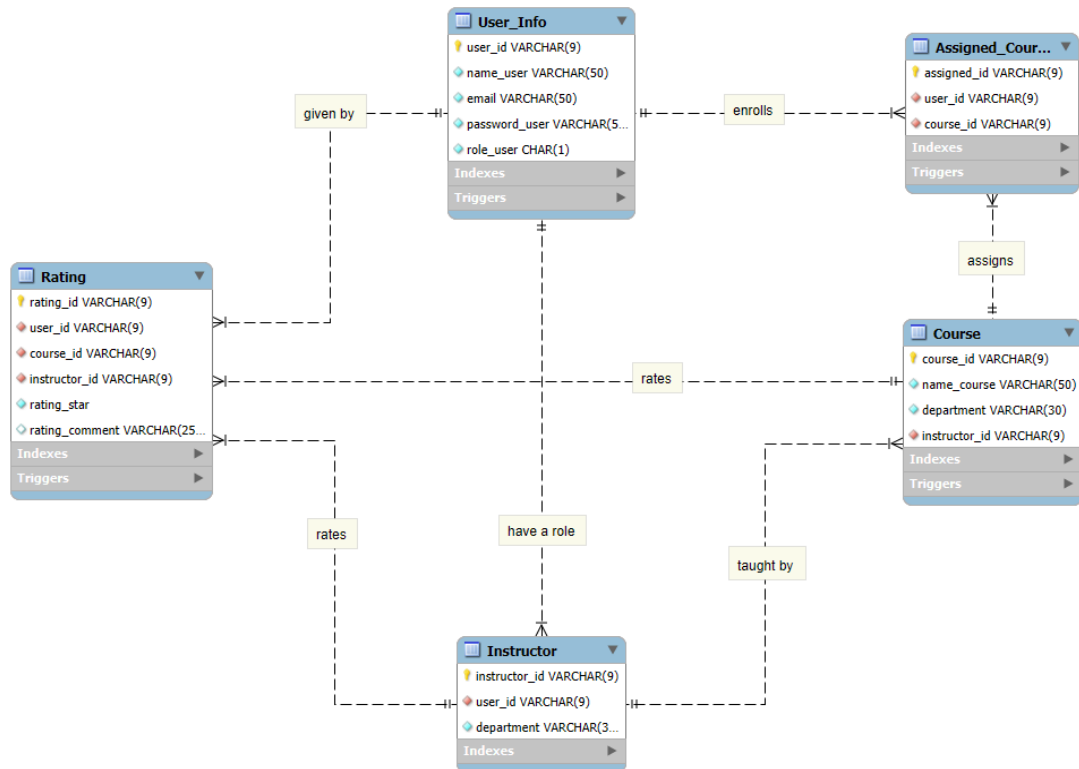


Figure 1: ERD Diagram for Database

- The little key symbol shows that attribute is the Primary Key for that entity.
- Red symbol shows that attribute is the Foreign Key for that entity.

3. IMPLEMENTATION

3.1. Execution Procedure and SQL Scripts

1- Database Setup

- **Dropping Existing Database:** If a database named *Rating DB* already exists, it must be dropped to avoid conflicts.

```
drop schema if exists RatingDB;
```

Figure 2: SQL command to remove any pre-existing database schema named *Rating DB*.

- **Creating a New Database:** Create the main database schema.

```
CREATE DATABASE RatingDB;
```

Figure 3: SQL command to create a new database for the project.

- **Set Active Database:** Specify *Rating DB* as the active database.

```
USE RatingDB;
```

Figure 4: SQL command to set the active database for subsequent operations.

2- Table Creation

The database schema includes several tables with dependencies. These should be created in the following order to resolve any relationships:

***User_Info* Table:**

```
CREATE TABLE User_Info (  
    user_id VARCHAR(9) PRIMARY KEY,  
    name_user VARCHAR(50) NOT NULL,  
    email VARCHAR(50) UNIQUE NOT NULL,  
    password_user VARCHAR(50) NOT NULL,  
    role_user CHAR(1) NOT NULL  
);
```

Figure 5: SQL command to create the *User_Info* table.

***Instructor* Table:**

```
CREATE TABLE Instructor (  
    instructor_id VARCHAR(9) PRIMARY KEY,  
    user_id VARCHAR(9) NOT NULL,  
    department VARCHAR(30) NOT NULL,  
    FOREIGN KEY (user_id) REFERENCES User_Info(user_id)  
);
```

Figure 6: SQL command to create the *Instructor* table.

***Course* Table:**

```
CREATE TABLE Course (  
    course_id VARCHAR(9) PRIMARY KEY,  
    name_course VARCHAR(50) NOT NULL,  
    department VARCHAR(30) NOT NULL,  
    instructor_id VARCHAR(9) NOT NULL,  
    FOREIGN KEY (instructor_id) REFERENCES Instructor(instructor_id)  
);
```

Figure 7: SQL command to create the *Course* table.

***Assigned_Course* Table:**

```
CREATE TABLE Assigned_Course (  
    assigned_id VARCHAR(9) PRIMARY KEY,  
    user_id VARCHAR(9) NOT NULL,  
    course_id VARCHAR(9) NOT NULL,  
    FOREIGN KEY (user_id) REFERENCES User_Info(user_id),  
    FOREIGN KEY (course_id) REFERENCES Course(course_id)  
);
```

Figure 8: SQL command to create the *Assigned_Course* table.

Rating Table:

```
CREATE TABLE Rating (  
    rating_id VARCHAR(9) PRIMARY KEY,  
    user_id VARCHAR(9) NOT NULL,  
    course_id VARCHAR(9) NOT NULL,  
    instructor_id VARCHAR(9) NOT NULL,  
    rating_star INT NOT NULL CHECK (rating_star BETWEEN 1 AND 5),  
    rating_comment VARCHAR(255),  
    FOREIGN KEY (user_id) REFERENCES User_Info(user_id),  
    FOREIGN KEY (course_id) REFERENCES Course(course_id),  
    FOREIGN KEY (instructor_id) REFERENCES Instructor(instructor_id)  
);
```

Figure 9: SQL command to create the *Rating* table.

3- Sample Data Insertion

Populate the tables with sample data. Insertions must follow the order of table creation:

User_Info Table:

```
INSERT INTO User_Info (user_id, name_user, email, password_user, role_user) VALUES  
( 'U001', 'Ezgi Su Bilal', 'bialele@mef.edu.tr', 'bubirsifredir123', 'S'),  
( 'U002', 'Elif İlayda Güntürk', 'ilaydaofmidgard@gmail.com', 'püskevit', 'S'),  
( 'U003', 'Charlie Brown', 'charlie@example.com', 'password123', 'S'),  
( 'U004', 'Bruce Wayne', 'wayne@waynecorps.com', 'password123', 'S'),  
( 'U005', 'Ethan Hunt', 'ethan@example.com', 'password123', 'S'),  
( 'U006', 'Pedro Pascal', 'pedro@example.com', 'password123', 'S'),  
( 'U007', 'Astarion Ancunin', 'bg3@example.com', 'password123', 'S'),  
( 'U008', 'Hannah Davis', 'hannah@example.com', 'password123', 'S'),  
( 'U009', 'Ian Curtis', 'ian@example.com', 'password123', 'S'),  
( 'U010', 'Julia Roberts', 'julia@example.com', 'password123', 'S'),  
( 'U011', 'Kevin Hart', 'kevin@example.com', 'password123', 'S'),  
( 'U012', 'Laura Palmer', 'laura@example.com', 'password123', 'S'),  
( 'U013', 'Michael Scott', 'michael@example.com', 'password123', 'S'),  
( 'U014', 'Nina Simone', 'nina@example.com', 'password123', 'S'),  
( 'U015', 'Oscar Wilde', 'oscar@example.com', 'password123', 'S'),  
( 'U016', 'Dr. Osamu Dazai ', 'osamu.dazai@example.com', 'securepass', 'I'),  
( 'U017', 'Dr. Utku Koç', 'koc.utku@example.com', 'securepass', 'I'),  
( 'U018', 'Dr. Alan Turing', 'alan.turing@example.com', 'securepass', 'I'),  
( 'U019', 'Dr. Grace Hopper', 'grace.hopper@example.com', 'securepass', 'I'),  
( 'U020', 'Prof. Özgür Özlük', 'ozgur@ozluk.com', 'securepass', 'I'),  
( 'U021', 'Dr. Tim Berners-Lee', 'tim.berners@example.com', 'securepass', 'I');
```

Figure 10: SQL command to insert sample data into the *User_Info* table.

Instructor Table:

```
INSERT INTO Instructor (instructor_id, user_id, department) VALUES
('I001', 'U016', 'Computer Science'),
('I002', 'U017', 'Mathematics'),
('I003', 'U018', 'Artificial Intelligence'),
('I004', 'U019', 'Software Engineering'),
('I005', 'U020', 'Data Science'),
('I006', 'U021', 'Web Development');
```

*Figure 11: SQL command to insert sample data into the *Instructor* table.*

Course Table:

```
INSERT INTO Course (course_id, name_course, department, instructor_id) VALUES
('C001', 'Introduction to Programming', 'Computer Science', 'I001'),
('C002', 'Simulation', 'Mathematics', 'I002'),
('C003', 'Machine Learning Basics', 'Artificial Intelligence', 'I003'),
('C004', 'Software Design Patterns', 'Software Engineering', 'I004'),
('C005', 'Data Visualization', 'Data Science', 'I005'),
('C006', 'Web Development Essentials', 'Web Development', 'I006'),
('C007', 'Database Systems', 'Computer Science', 'I001'),
('C008', 'Linear Algebra', 'Mathematics', 'I002'),
('C009', 'Deep Learning Fundamentals', 'Artificial Intelligence', 'I003'),
('C010', 'Agile Software Development', 'Software Engineering', 'I004'),
('C011', 'Big Data Analytics', 'Data Science', 'I005'),
('C012', 'Advanced Front-End Development', 'Web Development', 'I006');
```

*Figure 12: SQL command to insert sample data into the *Course* table.*

Assigned_Course Table:

```
INSERT INTO Assigned_Course (assigned_id, user_id, course_id) VALUES
('A001', 'U001', 'C001'),
('A002', 'U002', 'C002'),
('A003', 'U003', 'C003'),
('A004', 'U004', 'C004'),
('A005', 'U005', 'C005'),
('A006', 'U006', 'C006'),
('A007', 'U007', 'C001'),
('A008', 'U008', 'C002'),
('A009', 'U009', 'C003'),
('A010', 'U010', 'C004'),
('A011', 'U011', 'C005'),
('A012', 'U012', 'C006'),
('A013', 'U013', 'C001'),
('A014', 'U014', 'C003'),
('A015', 'U015', 'C004');
```

*Figure 13: SQL command to insert sample data into the *Assigned_Course* table.*

Rating Table:

```
INSERT INTO Rating (rating_id, user_id, course_id, instructor_id, rating_star, rating_comment) VALUES
('R001', 'U007', 'C001', 'I001', 3, 'Great course! Learned a lot. Hate the instructor. '),
('R002', 'U002', 'C002', 'I002', 4, 'Challenging but rewarding. '),
('R003', 'U003', 'C003', 'I003', 5, 'Well-structured and informative. '),
('R004', 'U004', 'C004', 'I004', 2, NULL),
('R005', 'U005', 'C005', 'I005', 5, 'The instructor was fantastic! '),
('R006', 'U006', 'C006', 'I006', 4, 'Clear and concise explanations. '),
('R007', 'U007', 'C001', 'I001', 4, 'Good introduction to programming. '),
('R008', 'U008', 'C002', 'I002', 3, 'A bit difficult but worth it. '),
('R009', 'U009', 'C003', 'I003', 5, 'Excellent course on AI. '),
('R010', 'U010', 'C004', 'I004', 4, 'Loved the practical examples. '),
('R011', 'U011', 'C005', 'I005', 5, 'Highly recommend this course. '),
('R012', 'U012', 'C006', 'I006', 1, NULL),
('R013', 'U013', 'C001', 'I001', 5, 'Very engaging and helpful. '),
('R014', 'U014', 'C003', 'I003', 5, 'Interesting and well-explained. '),
('R015', 'U015', 'C004', 'I004', 5, 'Really enjoyed the design patterns. ');
```

Figure 14: SQL command to insert sample data into the *Assigned_Course* table.

4-Procedures and Triggers

Prevent Duplicate Ratings:

```
DELIMITER $$

CREATE TRIGGER PreventDuplicateRatings
BEFORE INSERT ON Rating
FOR EACH ROW
BEGIN
    IF EXISTS (
        SELECT 1
        FROM Rating
        WHERE user_id = NEW.user_id
              AND course_id = NEW.course_id
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A user cannot rate the same course more than once.';
    END IF;
END$$
```

Figure 15: SQL trigger to prevent duplicate ratings by the same user for a single course.

Auto-Increment for User IDs:

```
-- Auto-Incremental Trigger for User_Info Table

DELIMITER $$

• CREATE TRIGGER auto_increment_user_id
  BEFORE INSERT ON User_Info
  FOR EACH ROW
  BEGIN
    -- Get the maximum user_id from the table and increment it
    DECLARE max_id INT;
    SET max_id = (SELECT COALESCE(MAX(CAST(SUBSTRING(user_id, 2) AS UNSIGNED)), 0) FROM User_Info);

    -- Generate the new ID
    SET NEW.user_id = CONCAT('U', LPAD(max_id + 1, 3, '0'));
  END$$

DELIMITER ;
```

Figure 16: SQL trigger for auto-incrementing user IDs.

Prevent Invalid Ratings:

```
-- No rating for unassigned course
DELIMITER //

CREATE TRIGGER prevent_invalid_rating
  BEFORE INSERT ON Rating
  FOR EACH ROW
  BEGIN
    IF NOT EXISTS (
      SELECT 1
      FROM Assigned_Course
      WHERE user_id = NEW.user_id AND course_id = NEW.course_id
    ) THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'User is not assigned to this course!';
    END IF;
  END; //

DELIMITER ;
```

Figure 17: SQL trigger to prevent invalid ratings.

Assign User to a Course Procedure

```
CREATE PROCEDURE AssignUserToCourse(  
    IN userID VARCHAR(9),  
    IN courseID VARCHAR(9)  
)  
BEGIN  
    -- Check if the user is already assigned to the course  
    IF EXISTS (  
        SELECT 1  
        FROM Assigned_Course  
        WHERE user_id = userID AND course_id = courseID  
    ) THEN  
        SIGNAL SQLSTATE '45000'  
        SET MESSAGE_TEXT = 'User is already assigned to this course!';  
    ELSE  
        -- Assign the user to the course (assigned_id is auto-incremented by the trigger)  
        INSERT INTO Assigned_Course (user_id, course_id)  
        VALUES (userID, courseID);  
    END IF;  
END$$  
  
DELIMITER ;
```

Figure 18: Procedure to assign users to courses.

Add a New Course Procedure

```
-- Adding a New Course  
DELIMITER $$  
  
CREATE PROCEDURE AddCourse(  
    IN courseName VARCHAR(50),  
    IN departmentName VARCHAR(30),  
    IN instructorID VARCHAR(9)  
)  
BEGIN  
    -- Insert a new course; course_id will be auto-generated by the trigger  
    INSERT INTO Course (name_course, department, instructor_id)  
    VALUES (courseName, departmentName, instructorID);  
END$$  
  
DELIMITER ;
```

Figure 19: Procedure to add a new course.

3.2. Alignment of Design with Requirements

1. Meeting the Requirements

- **Reliable and Structured Feedback:**
 - The *Rating* table collects ratings (1–5) and comments, ensuring a robust feedback mechanism.
 - Ratings are tied to verified users (*user_id* in *User_Info*) to ensure authenticity.
- **Actionable Insights for Improvement:**
 - Courses are linked to departments and instructors via the *Course* and *Instructor* tables, enabling insights at multiple levels.
 - The *Rating* table supports aggregated data such as average scores, which can help institutions improve course quality and teaching.
- **Security and Anonymity:**
 - User IDs (*user_id*) and anonymized feedback (*rating_comment*) ensure privacy.
 - Triggers and constraints, such as the *PreventDuplicateRatings* trigger, ensure data integrity and prevent abuse.
- **Scalability:**
 - Auto-incrementing triggers for primary keys ensure the system can accommodate a growing user base.
 - Normalized tables eliminate redundancy, optimizing database performance for large-scale operations.

2. Adhering to Assumptions

- **Verified Users:**
 - Assumption: Only verified users (students or alumni) provide feedback.
 - Implementation: The *User_Info* table stores user roles (*role_user*), ensuring that only eligible users can submit ratings.

- **Honest and Unbiased Ratings:**
 - Assumption: Ratings reflect genuine experiences.
 - Implementation: Anonymity in feedback protects users from bias while submitting ratings.
- **Adoption by Institutions:**
 - Assumption: Institutions will adopt insights for continuous improvement.
 - Implementation: Tools for administrators to make data actionable.

3. Respecting Constraints

- **Limiting Feedback to Eligible Users:**
 - Constraint: Only students who have completed or are enrolled in a course can provide ratings.
 - Implementation: The *Assigned_Course* table links users to courses, ensuring that only eligible students can rate courses. The *prevent_invalid_rating* trigger enforces this constraint.
- **One Rating per Course per User:**
 - Constraint: A user can rate a course only once.
 - Implementation: The *PreventDuplicateRatings* trigger prevents duplicate entries in the *Rating* table.
- **Scalability:**
 - Constraint: The system must handle increased traffic as the user base grows.
 - Implementation: Efficient indexing and normalized tables ensure optimal performance.

4. Satisfying Business Rules

- **Every Course Belongs to a Department:**
 - Implementation: The *department* attribute in the *Course* table enforces this rule.
- **Only Eligible Students Can Rate Courses:**
 - Implementation: The *Assigned_Course* table ensures that users are linked to courses before rating.
- **Ratings Between 1 and 5:**
 - Implementation: The *rating_star* column in the *Rating* table is constrained to values between 1 and 5 using a *CHECK* constraint.
- **Minimum Ratings for Course Display:**
 - Implementation: Queries calculate and display average ratings only if a course has at least three ratings.
- **Instructor Access to Feedback:**
 - Implementation: Instructors can access anonymized feedback linked to their courses through SQL queries.

4. NUMERICAL EXAMPLE

1. AssignUserToCourse Procedure

Table 1: Assigned_Course table before AssignUserToCourse Procedure

| | assigned_id | user_id | course_id |
|---|-------------|---------|-----------|
| ▶ | A001 | U001 | C001 |
| | A002 | U002 | C002 |
| | A003 | U003 | C003 |
| | A004 | U004 | C004 |
| | A005 | U005 | C005 |
| | A006 | U006 | C006 |
| | A007 | U007 | C001 |
| | A008 | U008 | C002 |
| | A009 | U009 | C003 |
| | A010 | U010 | C004 |
| | A011 | U011 | C005 |
| | A012 | U012 | C006 |
| | A013 | U013 | C001 |
| | A014 | U014 | C003 |
| | A015 | U015 | C004 |
| | A016 | U016 | C002 |
| * | NULL | NULL | NULL |

Assigned_Course2 x

```
CALL AssignUserToCourse('U008', 'C003');
```

Figure 20: Assigning a new course C003 to U008

Table 2: After the CALL Assigned_Course procedure U008 was assigned to C003

| | assigned_id | user_id | course_id |
|---|-------------|---------|-----------|
| | A002 | U002 | C002 |
| | A003 | U003 | C003 |
| | A004 | U004 | C004 |
| | A005 | U005 | C005 |
| | A006 | U006 | C006 |
| | A007 | U007 | C001 |
| | A008 | U008 | C002 |
| | A009 | U009 | C003 |
| | A010 | U010 | C004 |
| | A011 | U011 | C005 |
| | A012 | U012 | C006 |
| | A013 | U013 | C001 |
| | A014 | U014 | C003 |
| | A015 | U015 | C004 |
| | A016 | U016 | C002 |
| | A017 | U008 | C003 |
| * | NULL | NULL | NULL |

2. Adding a New Course Procedure

Table 3: Course table before AddCourse Procedure

| | course_id | name_course | department | instructor_id |
|---|-----------|------------------------------|-------------------------|---------------|
| ▶ | C001 | Introduction to Programming | Computer Science | I001 |
| | C002 | Simulation | Mathematics | I002 |
| | C003 | Machine Learning Basics | Artificial Intelligence | I003 |
| | C004 | Software Design Patterns | Software Engineering | I004 |
| | C005 | Data Visualization | Data Science | I005 |
| | C006 | Web Development Essentials | Web Development | I006 |
| | C007 | Database Systems | Computer Science | I001 |
| | C008 | Linear Algebra | Mathematics | I002 |
| | C009 | Deep Learning Fundamentals | Artificial Intelligence | I003 |
| | C010 | Agile Software Development | Software Engineering | I004 |
| | C011 | Big Data Analytics | Data Science | I005 |
| | C012 | Advanced Front-End Develo... | Web Development | I006 |
| | C013 | Cloud Computing | Computer Science | I001 |
| ✱ | NULL | NULL | NULL | NULL |

```
CALL AddCourse('Using ChatGPT as a Virtual Collabrator', 'Artificial Intelligence', 'I003');
```

Figure 21: Creating a new course from instructor I003

Table 4: After CALL AddCourse procedure U008 to C003.

| | course_id | name_course | department | instructor_id |
|---|-----------|--------------------------------|-------------------------|---------------|
| ▶ | C001 | Introduction to Programming | Computer Science | I001 |
| | C002 | Simulation | Mathematics | I002 |
| | C003 | Machine Learning Basics | Artificial Intelligence | I003 |
| | C004 | Software Design Patterns | Software Engineering | I004 |
| | C005 | Data Visualization | Data Science | I005 |
| | C006 | Web Development Essentials | Web Development | I006 |
| | C007 | Database Systems | Computer Science | I001 |
| | C008 | Linear Algebra | Mathematics | I002 |
| | C009 | Deep Learning Fundamentals | Artificial Intelligence | I003 |
| | C010 | Agile Software Development | Software Engineering | I004 |
| | C011 | Big Data Analytics | Data Science | I005 |
| | C012 | Advanced Front-End Develo... | Web Development | I006 |
| | C013 | Cloud Computing | Computer Science | I001 |
| | C014 | Using ChatGPT as a Virtual ... | Artificial Intelligence | I003 |
| ✱ | NULL | NULL | NULL | NULL |

3. PreventDuplicateRatings Trigger

```
CREATE TRIGGER PreventDuplicateRatings
BEFORE INSERT ON Rating
FOR EACH ROW
BEGIN
    IF EXISTS (
        SELECT 1
        FROM Rating
        WHERE user_id = NEW.user_id
              AND course_id = NEW.course_id
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A user cannot rate the same course more than once.';
    END IF;
END$$

DELIMITER ;

INSERT INTO Rating (user_id, course_id, instructor_id, rating_star, rating_comment) VALUES
('U007', 'C001', 'I001', 3, 'Great course! Learned a lot. Hate the instructor.');
```

Figure 22: After trying to add a new rating on a course that already has been rated before.

Error Code: 1644. A user cannot rate the same course more than once.

Figure 23: Error of the code

5. CONCLUSION

The problem of the subject of this project is the lack of an organized way for formation of an Online Course Rating System to facilitate students' feedback. It covers the process of designing a relational database, development of SQL queries for CRUD operations, and following the ethical and technical standards for implementation. The project was rather useful to all of us to have a look at data consistency, normalization, and more specific SQL features which all in all helped us to develop our database management abilities.

5.1. Broad Impact

Engineering solutions, such as the Online Course Rating System, have significant implications:

Environmental: Regarding the technological implications, the implementation of the system means the exclusion of paper-based feedback that contributes to environmental problems and, therefore, to goals and objectives related to sustainability. It assists academic *institutions* to cut their carbon footprints through the processes of digitization.

Economic: To that end, the system contributes greatly to this aspect by minimizing other costs such as dropped courses, and repeated enrollments due to lengthened course selection processes. Institutions are also the ones that are likely to benefit from enhanced efficiency in Resource management.

Societal: This feedback system ensures that numerous students and institutions have trust and transparency in their educational systems and therefore helps to improve those standards and education decision-making for students.

Cultural: The adaptive design complies with international practices and cultural homogenization, as well as having a positive influence on learning disparities.

Health: Knowing what courses to take lowers scholars' academic strain and, consequently, boosts the former's quality of life.

Legal: Being compliant with the rules of privacy like GDPR is also good for managing users' data safely and protecting organizations from probable legal responsibilities.

Security: Effective client protection mechanisms include; encrypting the frequencies of the systems, and restricting access to some of the systems through their role.

5.2. Life-Long Learning

This endeavor highlighted the continued necessity of professional learning in practice in particular. The following were important areas that needed further knowledge:

Advanced SQL Features: To create good database operations such as triggers and procedures features were mandatory. In order to find information about these areas, the tutorials and the official material were analyzed.

Database Security Practices: To understand encryption, access control and the ethical principles for safe systems, it was important to search for the material and works written by scholars, as well as standards and practices for the IT industry.

Data Analysis Techniques: By using samples from trustworthy sites and employing them during testing several times, it was possible to develop the practical knowledge of query optimization and explore the possibilities of SQL functions.

Moreover, an awareness of the most recent developments in technology and issues in the field was required to comprehend current topics, like current trends in instructional technology. Data was collected from professional workshops, SQL community forums, and scientific papers that are relevant to this topic. This procedure ensured a vital approach to both the creation and analysis of projects.

5.3. Professional and Ethical Responsibilities of Engineers

Therefore, professionalism and ethical considerations were important throughout the exercise. Therefore, based on the international ethical standards user privacy and data security were

considered as the top priority through ensuring anonymity of the users and securing their sensitive information. Through documentation of all design decisions and project events, maintainability was ensured, which promoted openness, and ensured an understanding of the system capabilities. In addition, the problem of openness and availability of the system for people with impairments proves its value once again.

The project also met moral requirements enshrined under the ACM Code of Ethics which focuses more on accountability, privacy, and justice. He or she ensured that every decision taken was done with integrity and keeping every party in mind.

5.4. Contemporary Issues/Future of Industry

The main idea of the project was based on the existing issues in educational technologies. Occasionally, or specifically when processing informational data it deployed and used present day methods such as SQL for effective manipulation. More innovative advancements are expected to greatly extend the realms of comparable systems as advancement persists in future.

Big data analytics, for instance, may provide granular insights into the feedback on courses provided thus helping organizations identify areas of strength, weakness, opportunity, and threats. Besides easing up the moderation of the input submitted to forums, the application of artificial intelligence has immense possibilities within personalized recommendations to students on courses to take. Online lecture observation and feedback may be attained through IoT during lectures allowing teachers to receive instant feedback about their classroom delivery. Conventional database queries that may take a long time to execute due to the many entries involved may be handled efficiently through quantum computing, which promises efficient execution of data sizes. Further, interactive interfaces could be changed significantly to enable easy interaction with instructional information at the nanoscale.

Such improvements will create wiser and more efficient systems during the next ten to fifty years of development. These advancements are expected to reshape the educational field by developing the learning process, improving decision-making mechanisms, and minimizing administrative work.

REFERENCES

1. OpenAI. (2024). ChatGPT [Large language model]. <https://chatgpt.com>