Bangladesh Civic & Learning Platform: A Database Management System

Student Name: Arfan Hosan Ovi
ID: 201002487

A project submitted in partial fulfillment of the requirements for the course

CSE-210: Database Lab

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GREEN UNIVERSITY OF BANGLADESH

2025

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Project Background

In Bangladesh, students, citizens, government, politics, and NGO activities typically operate in silos. This project aims to create a centralized, role-based platform where all stakeholders can collaborate effectively through a well-designed database system. The platform integrates learning management, civic engagement, project collaboration, and community interaction in a single system.

1.2 Objectives

- Design a normalized database schema for a multi-role platform
- Implement efficient data relationships and constraints
- Ensure data integrity through proper normalization
- Create SQL queries for various platform operations
- Develop a comprehensive database system for educational and civic activities

1.3 Scope

- Database design for a web-based system
- Role-based data access design
- Multi-module database integration (Learning, Civic, Community, Campaigns)
- Support for multiple user roles: Admin, Student, Citizen, Government, NGO
- Implementation of complex relationships and constraints

Chapter 2

Database Design

2.1 ER Diagram Concept

The Entity-Relationship diagram represents the logical structure of the database with tables organized into four modules:

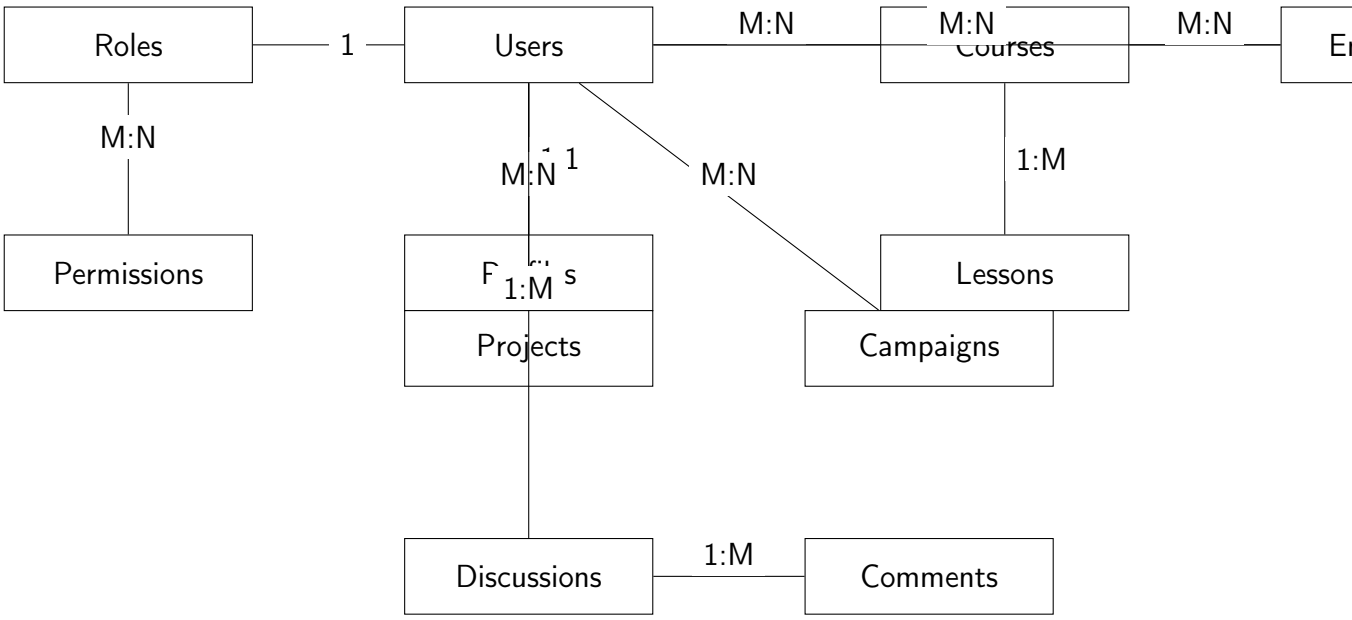

Figure 2.1: Conceptual Entity-Relationship Diagram of the Civic & Learning Platform

2.2 Database Schema

The database follows a modular design approach:

2.2.1 User & Role Base Module

- roles: Defines user roles (Admin, Student, Citizen, etc.)

- permissions: Specifies system permissions
- role_permissions: Maps permissions to roles
- users: Stores user authentication details
- profiles: Contains user profile information

2.2.2 Learning Module

- courses: Manages educational courses
- lessons: Contains course lessons
- enrollments: Tracks course enrollments
- assignments: Manages course assignments
- submissions: Stores assignment submissions
- quizzes: Manages course quizzes
- quiz_attempts: Tracks quiz attempts

2.2.3 Civic Projects & Campaigns Module

- projects: Manages civic projects
- project_participants: Tracks project participation
- project_updates: Stores project updates
- campaigns: Manages campaigns
- campaign_participation: Tracks campaign participation
- donations: Manages donation records

2.2.4 Community & Interaction Module

- discussions: Manages discussion threads
- comments: Stores comments on discussions
- votes: Tracks voting on content
- reactions: Manages user reactions to content
- events: Manages community events
- event_participants: Tracks event participation

2.3 Key SQL Implementation

2.3.1 Database Creation

```sql
CREATE DATABASE civic_learning_db;
USE civic_learning_db;

```

Listing 2.1: Database Creation

2.3.2 Table Creation Examples

```sql
CREATE TABLE roles (
    role_id INT PRIMARY KEY AUTO_INCREMENT,
    role_name VARCHAR(50) NOT NULL UNIQUE,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
   CURRENT_TIMESTAMP
);

```

Listing 2.2: Roles Table Creation

```sql
CREATE TABLE users (
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    role_id INT NOT NULL,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
   CURRENT_TIMESTAMP,
    FOREIGN KEY (role_id) REFERENCES roles(role_id)
);

```

Listing 2.3: Users Table Creation

```sql
CREATE TABLE profiles (
    profile_id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL UNIQUE,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    date_of_birth DATE,
    phone VARCHAR(20),
    address TEXT,
    bio TEXT,
    profile_picture VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
   CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE
   CASCADE
);

```

Listing 2.4: Profiles Table Creation

```sql
CREATE TABLE courses (
    course_id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    instructor_id INT NOT NULL,
    start_date DATE,
    end_date DATE,
    status ENUM('active', 'inactive', 'completed') DEFAULT 'active',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (instructor_id) REFERENCES users(user_id)
);
```

Listing 2.5: Courses Table Creation

```sql
CREATE TABLE enrollments (
    enrollment_id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    course_id INT NOT NULL,
    enrollment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status ENUM('active', 'completed', 'dropped') DEFAULT 'active',
    progress DECIMAL(5,2) DEFAULT 0.00,
    UNIQUE KEY unique_enrollment (user_id, course_id),
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (course_id) REFERENCES courses(course_id) ON DELETE CASCADE
);
```

Listing 2.6: Enrollments Table Creation

```sql
CREATE TABLE projects (
    project_id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    creator_id INT NOT NULL,
    start_date DATE,
    end_date DATE,
    status ENUM('planning', 'active', 'completed', 'cancelled') DEFAULT 'planning',
    target_fund DECIMAL(15,2),
    current_fund DECIMAL(15,2) DEFAULT 0.00,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (creator_id) REFERENCES users(user_id)
);
```

Listing 2.7: Projects Table Creation

2.4 Relationships

- One-to-One: users âĘŤ profiles

- One-to-Many: users âĘŤ courses (instructor), users âĘŤ projects (creator)
- Many-to-Many: users âĘŤ courses (enrollments), users âĘŤ projects (participants), users âĘŤ campaigns (participation)

2.5 Normalization

The database is normalized to Third Normal Form (3NF) to eliminate data redundancy and ensure data integrity. All tables have proper primary keys, foreign key relationships, and follow normalization principles.

Chapter 3

SQL Queries

3.1 Data Insertion Queries

```sql
-- Insert roles
INSERT INTO roles (role_name, description) VALUES
('Admin', 'System administrator with full access'),
('Student', 'Can enroll in courses and submit assignments'),
('Citizen', 'Can participate in civic activities and discussions'),
('Government', 'Can create and manage civic projects'),
('NGO', 'Can create campaigns and manage volunteers');

-- Insert permissions
INSERT INTO permissions (permission_name, description) VALUES
('create_course', 'Permission to create new courses'),
('enroll_course', 'Permission to enroll in courses'),
('create_project', 'Permission to create civic projects'),
('join_project', 'Permission to join projects'),
('manage_users', 'Permission to manage user accounts');

-- Map permissions to roles
INSERT INTO role_permissions (role_id, permission_id) VALUES
(1, 1), (1, 2), (1, 3), (1, 4), (1, 5),  -- Admin has all permissions
(2, 2), (2, 4),                          -- Student can enroll and join
(3, 2), (3, 4),                          -- Citizen can enroll and join
(4, 1), (4, 3), (4, 4),                  -- Government can create courses and projects
(5, 1), (5, 3), (5, 4);                  -- NGO can create courses and projects

-- Insert a user
INSERT INTO users (email, password, role_id) VALUES
('admin@example.com', 'hashed_password_1', 1),
('student1@example.com', 'hashed_password_2', 2),
('citizen1@example.com', 'hashed_password_3', 3),
('gov_official@example.com', 'hashed_password_4', 4),
('ngo_worker@example.com', 'hashed_password_5', 5);

```

```sql
-- Insert profiles
INSERT INTO profiles (user_id, first_name, last_name, phone,
    address) VALUES
(1, 'Admin', 'User', '+8801711111111', 'Dhaka, Bangladesh'),
(2, 'John', 'Doe', '+8801722222222', 'Chittagong, Bangladesh'),
(3, 'Jane', 'Smith', '+8801733333333', 'Sylhet, Bangladesh'),
(4, 'Robert', 'Ahmed', '+8801744444444', 'Rajshahi, Bangladesh'),
(5, 'Maria', 'Khan', '+8801755555555', 'Khulna, Bangladesh');

-- Insert a course
INSERT INTO courses (title, description, instructor_id, start_date
    , end_date) VALUES
('Introduction to Database Systems', 'Fundamentals of database
    management', 4, '2023-09-01', '2023-12-15'),
('Civic Engagement in Bangladesh', 'Understanding civic
    responsibilities', 5, '2023-10-01', '2024-01-15');

-- Insert enrollments
INSERT INTO enrollments (user_id, course_id) VALUES
(2, 1), (3, 1), (2, 2), (3, 2), (4, 2), (5, 2);

-- Insert a project
INSERT INTO projects (title, description, creator_id, start_date,
    end_date, target_fund) VALUES
('Clean Dhaka Initiative', 'A project to clean and maintain public
     spaces in Dhaka', 4, '2023-11-01', '2024-06-30', 500000.00),
('Digital Literacy for Rural Areas', 'Providing digital education
    in rural communities', 5, '2023-12-01', '2024-08-31',
    750000.00);

```

Listing 3.1: Inserting Sample Data

3.2 Data Retrieval Queries

```sql
SELECT u.user_id, p.first_name, p.last_name, p.email
FROM users u
JOIN profiles p ON u.user_id = p.user_id
WHERE u.role_id = (SELECT role_id FROM roles WHERE role_name = '
    Student');

```

Listing 3.2: Query to Get All Students

```sql
SELECT c.title AS course_title,
       CONCAT(p.first_name, ' ', p.last_name) AS student_name,
       p.email,
       e.enrollment_date,
       e.status,
       e.progress
FROM enrollments e
JOIN users u ON e.user_id = u.user_id
JOIN profiles p ON u.user_id = p.user_id
JOIN courses c ON e.course_id = c.course_id
ORDER BY c.title, p.last_name;

```

Listing 3.3: Query to Get Course Enrollments

```sql
SELECT p.title,
       CONCAT(prof.first_name, ' ', prof.last_name) AS creator_name,
       p.start_date,
       p.end_date,
       p.status,
       p.target_fund,
       p.current_fund,
       COUNT(pp.user_id) AS participant_count
FROM projects p
JOIN users u ON p.creator_id = u.user_id
JOIN profiles prof ON u.user_id = prof.user_id
LEFT JOIN project_participants pp ON p.project_id = pp.project_id
WHERE p.status = 'active'
GROUP BY p.project_id
ORDER BY p.start_date;

```

Listing 3.4: Query to Get Active Projects with Participation Count

3.3 Data Update Queries

```sql
UPDATE profiles
SET phone = '+8801712345678', address = 'Dhaka, Bangladesh'
WHERE user_id = 1;

```

Listing 3.5: Update User Profile

```sql
UPDATE enrollments
SET progress = 75.50, status = 'active'
WHERE user_id = 2 AND course_id = 1;

```

Listing 3.6: Update Course Progress

```sql
UPDATE projects
SET current_fund = current_fund + 5000.00
WHERE project_id = 1;

INSERT INTO donations (project_id, donor_id, amount, donation_date)
VALUES (1, 2, 5000.00, NOW());

```

Listing 3.7: Record a Donation to a Project

3.4 Data Deletion Queries

```sql
DELETE FROM users WHERE user_id = 5;

```

Listing 3.8: Delete a User Account

```sql
DELETE FROM enrollments
WHERE user_id = 3 AND course_id = 1;

```

Listing 3.9: Remove a User from a Course

Chapter 4

Advanced Database Features

4.1 Views

```sql
CREATE VIEW user_details AS
SELECT u.user_id, u.email, u.is_active, u.created_at,
       p.first_name, p.last_name, p.phone, p.address,
       r.role_name, r.description AS role_description
FROM users u
JOIN profiles p ON u.user_id = p.user_id
JOIN roles r ON u.role_id = r.role_id;

```

Listing 4.1: Creating a View for User Details

```sql
CREATE VIEW course_statistics AS
SELECT c.course_id, c.title, c.instructor_id,
       CONCAT(p.first_name, ' ', p.last_name) AS instructor_name,
       COUNT(e.user_id) AS enrollment_count,
       AVG(e.progress) AS average_progress,
       SUM(CASE WHEN e.status = 'completed' THEN 1 ELSE 0 END) AS
   completed_count
FROM courses c
JOIN users u ON c.instructor_id = u.user_id
JOIN profiles p ON u.user_id = p.user_id
LEFT JOIN enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id;

```

Listing 4.2: Creating a View for Course Statistics

4.2 Stored Procedures

```sql
DELIMITER //
CREATE PROCEDURE RegisterUser(
    IN p_email VARCHAR(255),
    IN p_password VARCHAR(255),
    IN p_role_name VARCHAR(50),
    IN p_first_name VARCHAR(100),
    IN p_last_name VARCHAR(100),
    IN p_phone VARCHAR(20),
    IN p_address TEXT
```

```
)
BEGIN
    DECLARE v_role_id INT;
    DECLARE v_user_id INT;

    -- Get role_id from role_name
    SELECT role_id INTO v_role_id FROM roles WHERE role_name =
    p_role_name;

    -- Insert into users table
    INSERT INTO users (email, password, role_id)
    VALUES (p_email, p_password, v_role_id);

    -- Get the last inserted user_id
    SET v_user_id = LAST_INSERT_ID();

    -- Insert into profiles table
    INSERT INTO profiles (user_id, first_name, last_name, phone,
    address)
    VALUES (v_user_id, p_first_name, p_last_name, p_phone,
    p_address);

    SELECT v_user_id AS new_user_id;
END //
DELIMITER ;
```

Listing 4.3: Stored Procedure for User Registration

```
DELIMITER //
CREATE PROCEDURE EnrollInCourse(
    IN p_user_id INT,
    IN p_course_id INT
)
BEGIN
    DECLARE v_enrollment_count INT;

    -- Check if already enrolled
    SELECT COUNT(*) INTO v_enrollment_count
    FROM enrollments
    WHERE user_id = p_user_id AND course_id = p_course_id;

    IF v_enrollment_count = 0 THEN
        -- Insert new enrollment
        INSERT INTO enrollments (user_id, course_id, status,
    progress)
        VALUES (p_user_id, p_course_id, 'active', 0.00);

        SELECT 'Enrollment successful' AS result;
    ELSE
        SELECT 'Already enrolled in this course' AS result;
    END IF;
END //
DELIMITER ;
```

Listing 4.4: Stored Procedure for Enrolling in a Course

4.3 Triggers

```
DELIMITER //
CREATE TRIGGER before_user_update
    BEFORE UPDATE ON users
    FOR EACH ROW
BEGIN
    SET NEW.updated_at = NOW();
END //
DELIMITER ;

```

Listing 4.5: Trigger to Update Timestamp

```
DELIMITER //
CREATE TRIGGER before_course_delete
    BEFORE DELETE ON courses
    FOR EACH ROW
BEGIN
    DECLARE enrollment_count INT;

    SELECT COUNT(*) INTO enrollment_count
    FROM enrollments
    WHERE course_id = OLD.course_id AND status = 'active';

    IF enrollment_count > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot delete course with active
   enrollments';
    END IF;
END //
DELIMITER ;

```

Listing 4.6: Trigger to Prevent Course Deletion with Active Enrollments

Chapter 5

Implementation and Testing

5.1 Database Implementation

The database was implemented using MySQL with the following steps:

1. Created the database schema with all tables
2. Established relationships using foreign keys
3. Implemented constraints for data integrity
4. Inserted sample data for testing
5. Created views, stored procedures, and triggers
6. Tested all functionalities with sample queries

5.2 Testing Scenarios

- User registration and authentication
- Course creation and enrollment
- Project creation and participation -based access control
- Data retrieval for reporting
- Data modification operations

5.3 Performance Considerations

- Indexes were created on frequently queried columns
- Proper data types were chosen to optimize storage

- Normalization was implemented to reduce redundancy
- Views were created for complex queries
- Stored procedures were used for common operations

Chapter 6

Conclusion

6.1 Summary

The Bangladesh Civic & Learning Platform database provides a robust foundation for an integrated platform that bridges education, civic engagement, and community collaboration. The normalized database design supports multiple user roles, complex relationships, and scalable operations. The implementation includes advanced database features like views, stored procedures, and triggers to enhance functionality and maintain data integrity.

6.2 Learning Outcomes

Through this project, I have gained practical experience in:

- Database design and normalization
- Entity-Relationship modeling
- SQL query implementation
- Creating complex relationships between tables
- Implementing constraints for data integrity
- Developing advanced database features (views, procedures, triggers)
- Testing and optimizing database performance

6.3 Challenges and Solutions

- Challenge: Designing a flexible schema for multiple user roles Solution: Used role-based access control pattern with separate roles and permissions tables

- Challenge: Managing many-to-many relationships Solution: Created junction tables like enrollments and project_participants
- Challenge: Ensuring data integrity Solution: Implemented foreign key constraints and appropriate cascading actions
- Challenge: Implementing complex business logic Solution: Used stored procedures and triggers to encapsulate logic

6.4 Future Enhancements

- Add full-text search capabilities for courses and projects
- Implement more advanced reporting and analytics features
- Add support for multimedia content in courses
- Implement real-time notifications using database events
- Add geographic data support for location-based services
- Implement data encryption for sensitive information

References

Bibliography

[1] Date, C. J. (2003). *An Introduction to Database Systems*. Addison-Wesley.

[2] MySQL Documentation. (2023). *MySQL 8.0 Reference Manual*. Oracle Corporation.

[3] Connolly, T. M., & Begg, C. E. (2015). *Database Systems: A Practical Approach to Design, Implementation, and Management*. Pearson Education.

[4] Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Database System Concepts* (7th ed.). McGraw-Hill.

Appendix A

Complete SQL Schema

Appendix B

Sample Data Insertion Script