Unicomtic Management System project report

Introduction / Abstract

The Unicomtic Management System is a simple desktop application developed to manage student-related academic information efficiently. It allows students to log in, view their timetables, and access essential academic details through a user-friendly interface. The system was built using C# and follows Object-Oriented Programming principles to ensure modularity and maintainability.

Problem Statement

There is a need for a digital system that allows students, staff, lecturers, and administrators to easily access their timetables and academic details based on their roles, while reducing administrative workload and enhancing data accuracy.

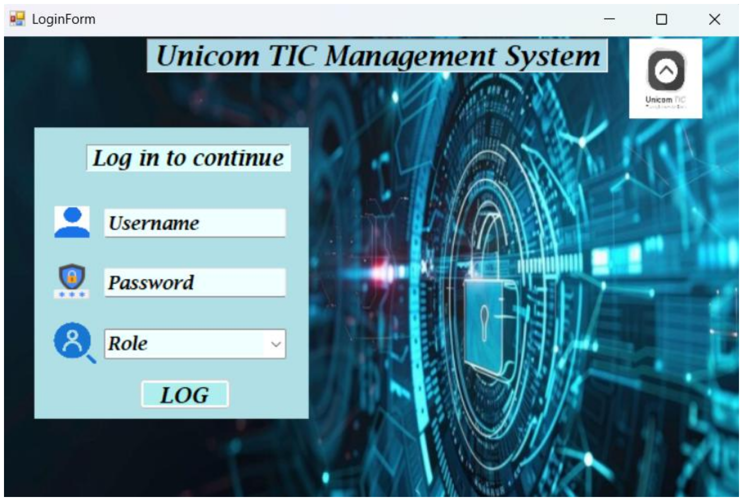
System Design

The Unicomtic Management System follows a modular design using Object-Oriented Programming. It is developed as a Windows Forms application in C# with a back-end SQL database for storing student and timetable data. The system includes separate forms and functionalities for each user role (e.g., student, admin,lecturer and staff).

Technologies Used

- **Programming Language**: C#
- **Framework**: .NET (Windows Forms Application)
- **Database**: SQLite
- **IDE**: Visual Studio
- **UI Design**: Windows Forms Designer
- **Data Access**: SQLiteConnection, SQLiteCommand (System.Data.SQLite)
- **Programming Concepts**: Object-Oriented Programming (OOP), Event-Driven Programming

User Login Form and connection between Login Form and Users Table



LoginForm

- Initially, the Admin can log in using the **role 'Admin'** with the **username 'admin'** and **password '123'**.

- When you click on the Username, Password, or Role text box, the placeholder text will disappear, allowing you to type or select the appropriate value.

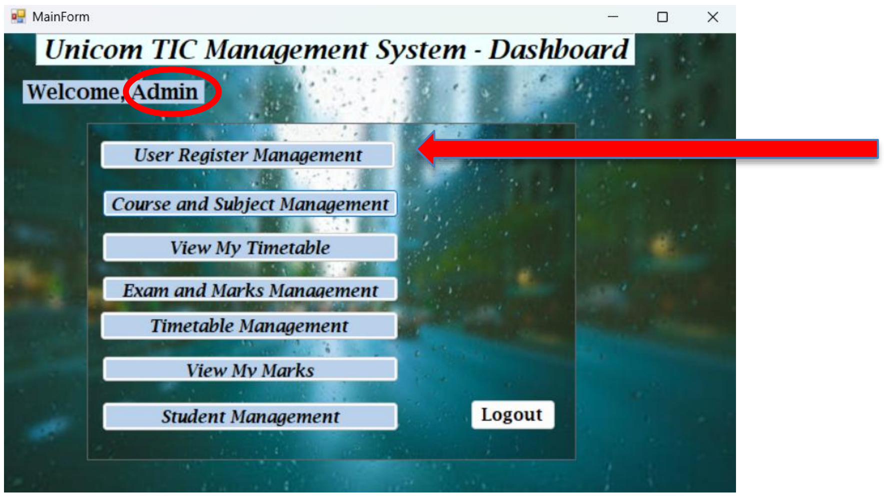
- After entering the **Username**, **Password**, and selecting the **Role**, the system checks the data against the **Users** table in the database.

If all three fields match a record correctly:

- A **"Login successful"** message will be displayed.
- You will be redirected to the **Main Dashboard** based on your role.

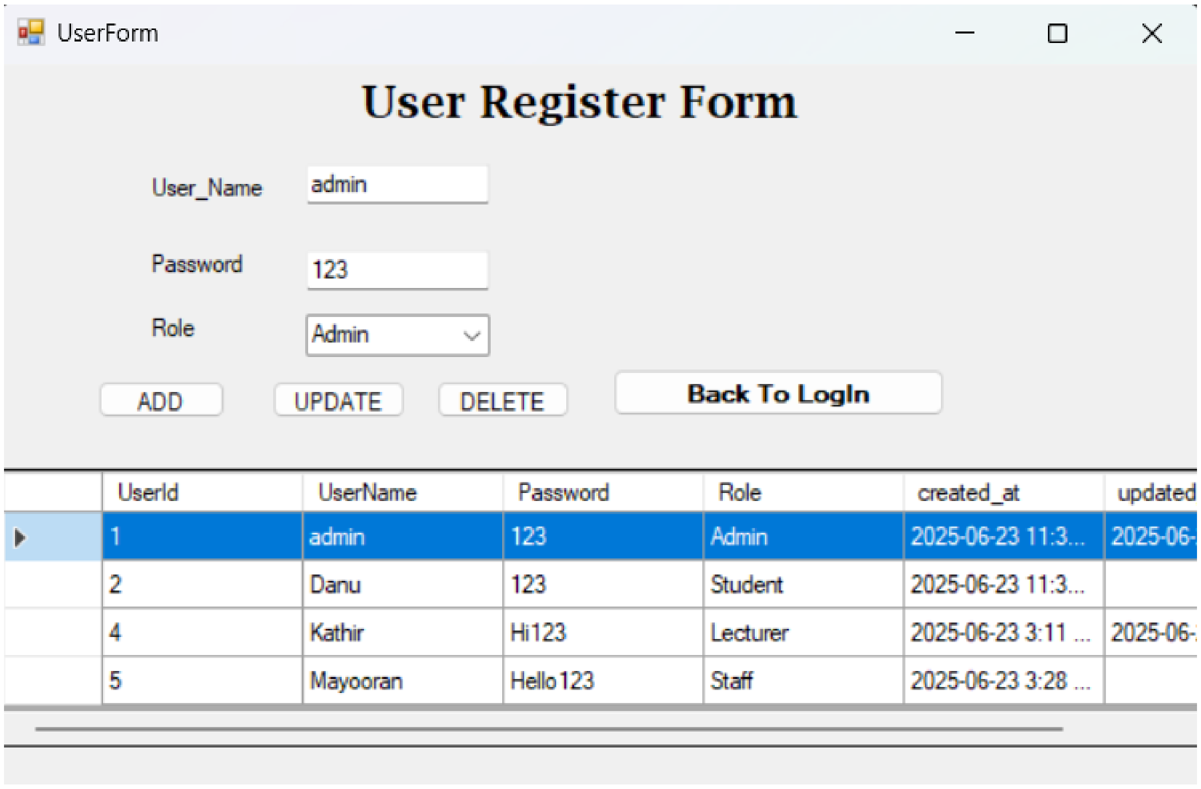
On the Dashboard:

- A **welcome message** will be shown with your **logged-in username**.
- If the user is an **Admin**, all buttons and features will be visible and accessible.
- If the user is **Staff**, **Lecturer**, or **Student**, access will be limited based on their role, and unauthorized buttons will be **hidden**.



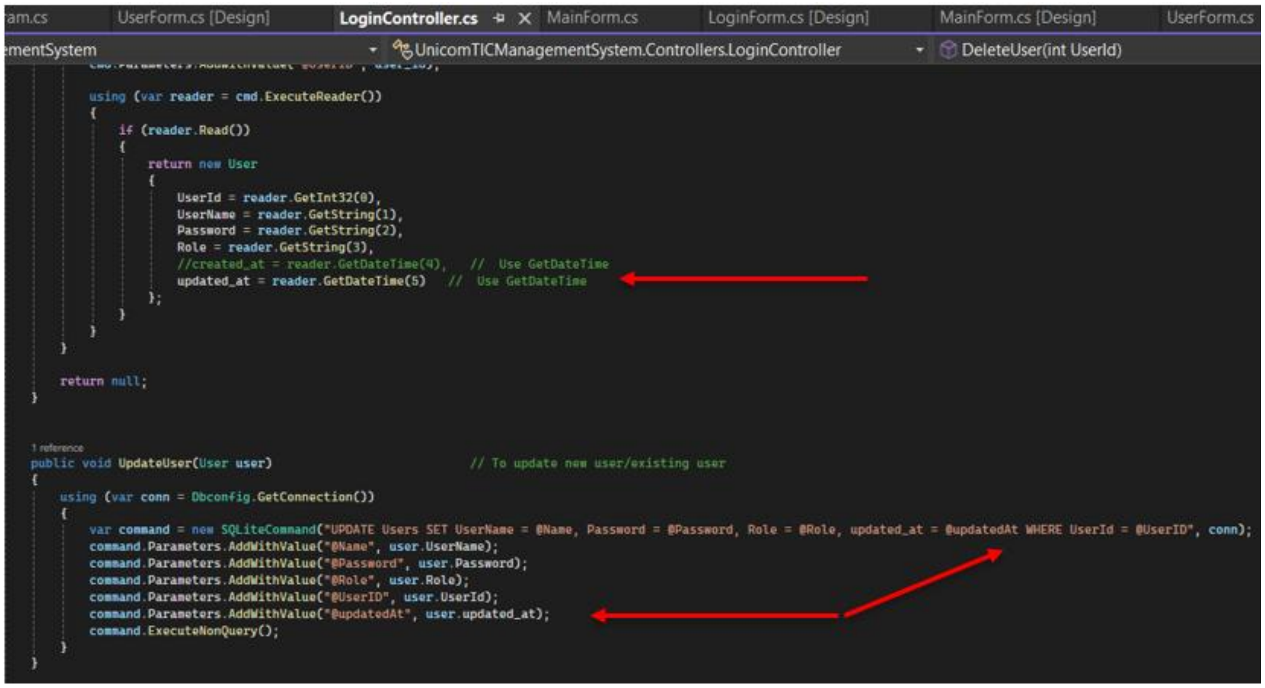
User Registration Management (Admin Only):

- Only the **Admin** has access to the **User Registration Management** feature.
- The Admin can **add**, **update**, and **delete** user accounts by managing their **username**, **password**, and **role.**
- The **created time** and **last updated time** of each user are displayed in the **DataGridView**.



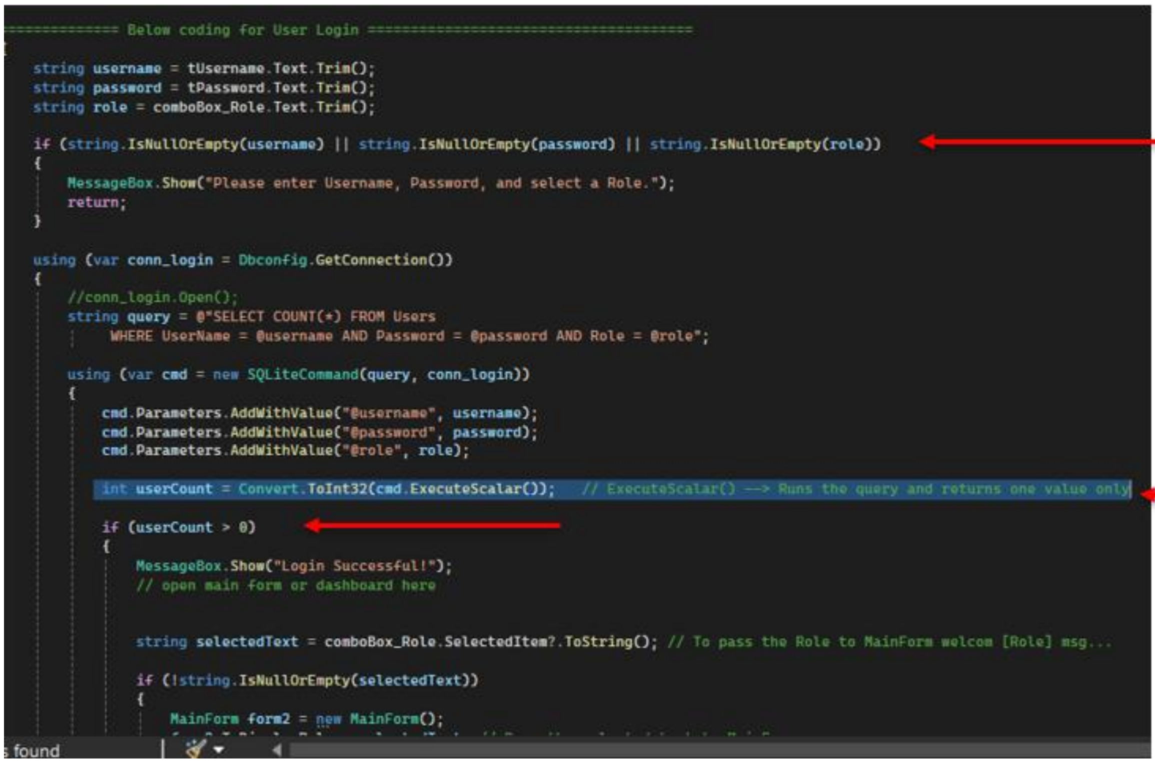
User_Name	admin
Password	123
Role	Admin

UserId	UserName	Password	Role	created_at	updated
1	admin	123	Admin	2025-06-23 11:3...	2025-06-
2	Danu	123	Student	2025-06-23 11:3...	
4	Kathir	Hi123	Lecturer	2025-06-23 3:11 ...	2025-06-
5	Mayooran	Hello123	Staff	2025-06-23 3:28 ...	



```
            using (var reader = cmd.ExecuteReader())
            {
                if (reader.Read())
                {
                    return new User
                    {
                        UserId = reader.GetInt32(0),
                        UserName = reader.GetString(1),
                        Password = reader.GetString(2),
                        Role = reader.GetString(3),
                        //created_at = reader.GetDateTime(4),   //  Use GetDateTime
                        updated_at = reader.GetDateTime(5)   //  Use GetDateTime
                    };
                }
            }
        }

        return null;
    }


    public void UpdateUser(User user)                          // To update new user/existing user
    {
        using (var conn = Dbconfig.GetConnection())
        {
            var command = new SQLiteCommand("UPDATE Users SET UserName = @Name, Password = @Password, Role = @Role, updated_at = @updatedAt WHERE UserId = @UserID", conn);
            command.Parameters.AddWithValue("@Name", user.UserName);
            command.Parameters.AddWithValue("@Password", user.Password);
            command.Parameters.AddWithValue("@Role", user.Role);
            command.Parameters.AddWithValue("@UserID", user.UserId);
            command.Parameters.AddWithValue("@updatedAt", user.updated_at);
            command.ExecuteNonQuery();
        }
    }
```

- Displaying the **'Created At'** and **'Updated At'** timestamps is one of my best features. It allows the admin to easily track when each user was created or last modified, directly within the DataGridView after any creation or update action.



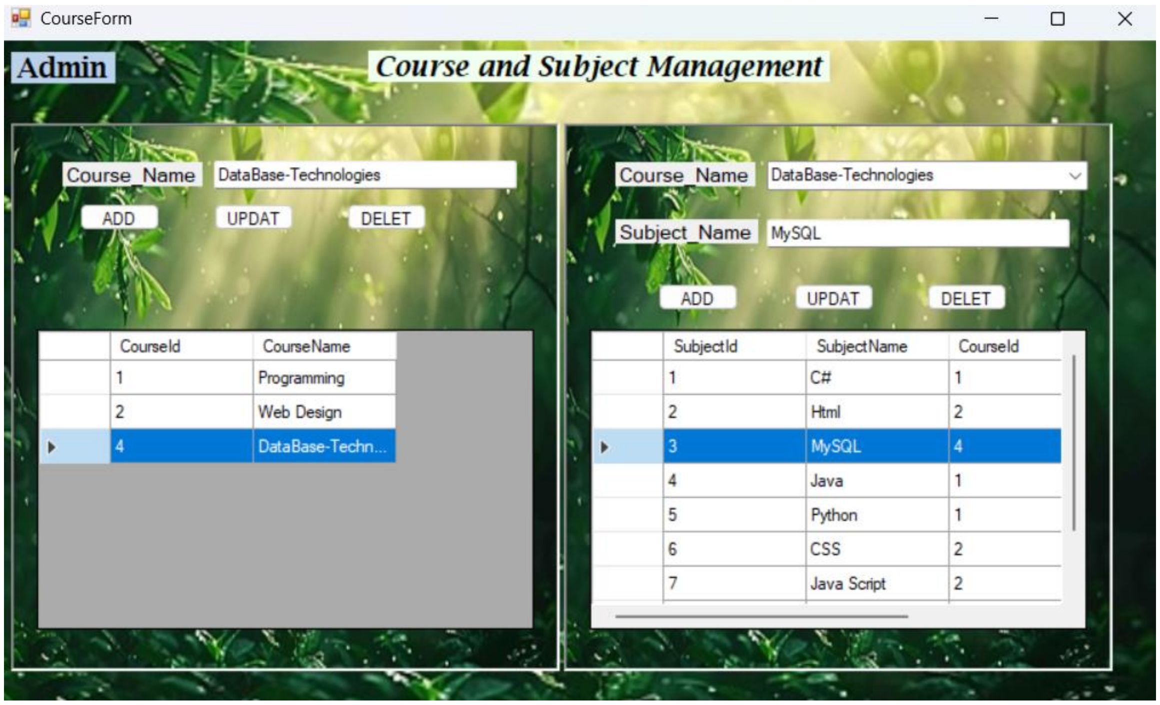
```
============= Below coding for User Login ======================================
{
    string username = tUsername.Text.Trim();
    string password = tPassword.Text.Trim();
    string role = comboBox_Role.Text.Trim();

    if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password) || string.IsNullOrEmpty(role))
    {
        MessageBox.Show("Please enter Username, Password, and select a Role.");
        return;
    }

    using (var conn_login = Dbconfig.GetConnection())
    {
        //conn_login.Open();
        string query = @"SELECT COUNT(*) FROM Users
            WHERE UserName = @username AND Password = @password AND Role = @role";

        using (var cmd = new SQLiteCommand(query, conn_login))
        {
            cmd.Parameters.AddWithValue("@username", username);
            cmd.Parameters.AddWithValue("@password", password);
            cmd.Parameters.AddWithValue("@role", role);

            int userCount = Convert.ToInt32(cmd.ExecuteScalar());   // ExecuteScalar() --> Runs the query and returns one value only

            if (userCount > 0)
            {
                MessageBox.Show("Login Successful!");
                // open main form or dashboard here


                string selectedText = comboBox_Role.SelectedItem?.ToString(); // To pass the Role to MainForm welcom [Role] msg...

                if (!string.IsNullOrEmpty(selectedText))
                {
                    MainForm form2 = new MainForm();
```

- @username, @password, and @role are **parameters** — they are replaced with the values typed in your form (tUsername.Text, tPassword.Text, comboBox_Role.Text).

- SQL checks how many records in the Users table **match exactly** those 3 values.

- Executes an SQL query that counts how many rows **match** the given username, password, and role.
  If count is 1 (or more) → it means a match exists.
  If count is 0 → invalid login.

Code	Meaning
ExecuteScalar()	Runs the query and returns **one value only**
userCount	Tells how many matching records exist
Convert.ToInt32( )	Converts the result (like 1) to an integer
if (userCount > 0)	Login success if **any match is found**



Course_Name: DataBase-Technologies

	CourseId	CourseName
	1	Programming
	2	Web Design
▶	4	DataBase-Techn...

Course_Name: DataBase-Technologies

Subject_Name: MySQL

	SubjectId	SubjectName	CourseId
	1	C#	1
	2	Html	2
▶	3	MySQL	4
	4	Java	1
	5	Python	1
	6	CSS	2
	7	Java Script	2

Course and Subject Access Control:

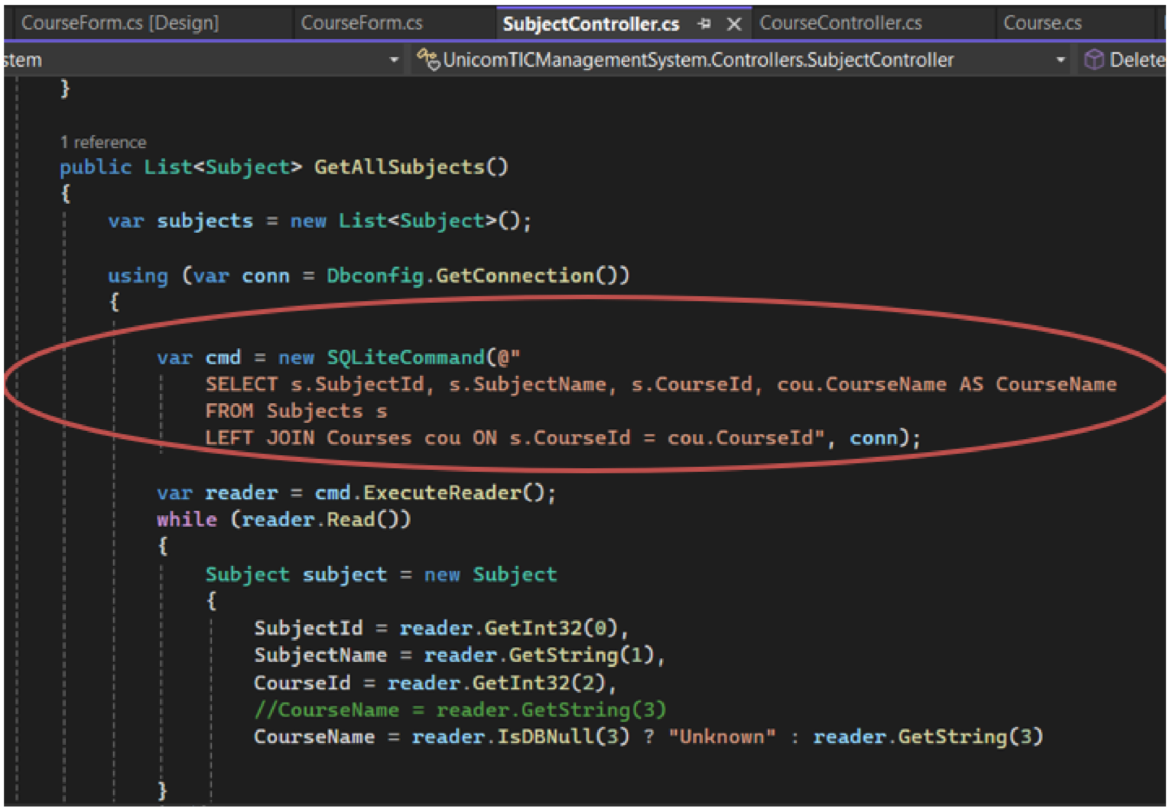
- **Admin** users have full access to the **Course** and **Subject** modules.

  They can **add**, **update**, and **delete** courses and related subjects.

- **Staff**, **Lecturers**, and **Students** have **read-only access**.

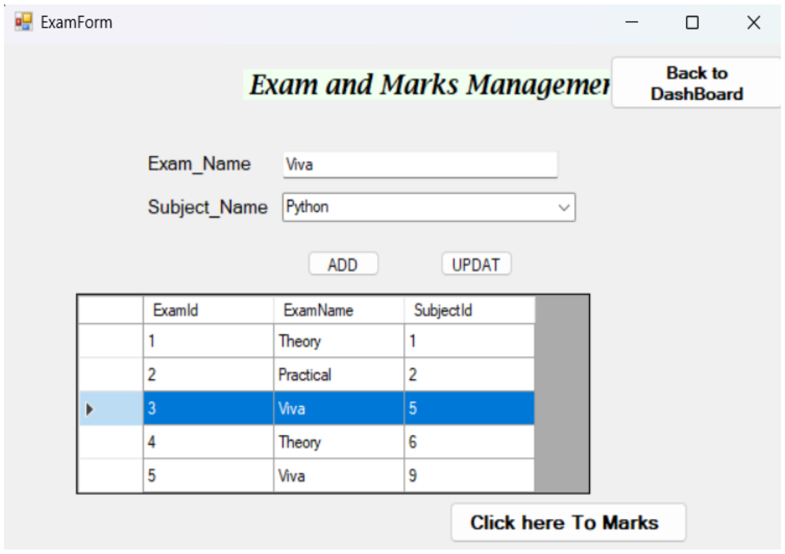
  They can **view the list of available courses and subjects** in the grid views.

  They **cannot add, update, or delete** any course or subject. **(visible hidden by code)**



```
    }

    public List<Subject> GetAllSubjects()
    {
        var subjects = new List<Subject>();

        using (var conn = Dbconfig.GetConnection())
        {

            var cmd = new SQLiteCommand(@"
                SELECT s.SubjectId, s.SubjectName, s.CourseId, cou.CourseName AS CourseName
                FROM Subjects s
                LEFT JOIN Courses cou ON s.CourseId = cou.CourseId", conn);

            var reader = cmd.ExecuteReader();
            while (reader.Read())
            {
                Subject subject = new Subject
                {
                    SubjectId = reader.GetInt32(0),
                    SubjectName = reader.GetString(1),
                    CourseId = reader.GetInt32(2),
                    //CourseName = reader.GetString(3)
                    CourseName = reader.IsDBNull(3) ? "Unknown" : reader.GetString(3)

            }
```

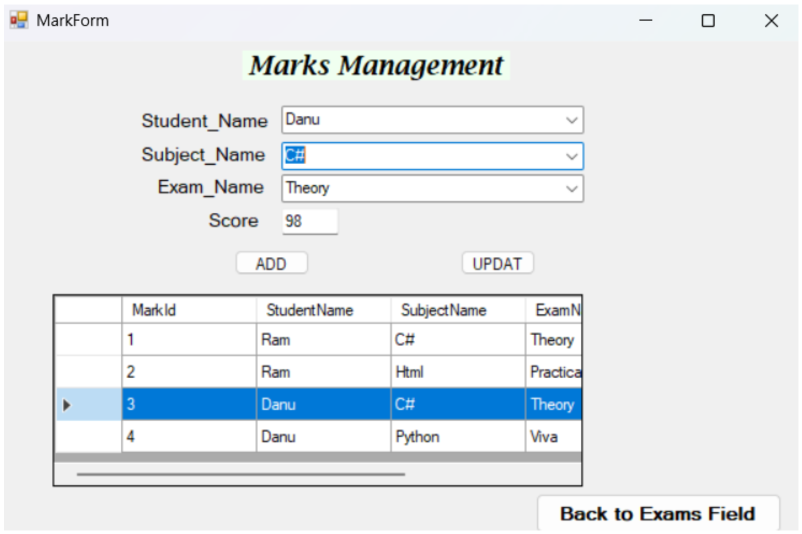
I used a **LEFT JOIN** to display the **Course Name** in the subject grid view by linking the **CourseId** foreign key from the Subjects table with the Courses table.



Exam and Marks Managemer

Exam_Name: Viva
Subject_Name: Python

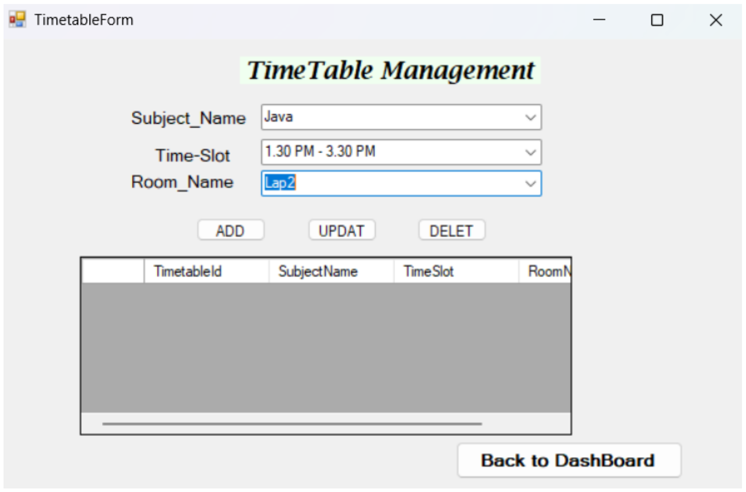
	ExamId	ExamName	SubjectId
	1	Theory	1
	2	Practical	2
▸	3	Viva	5
	4	Theory	6
	5	Viva	9



Marks Management

Student_Name: Danu
Subject_Name: C#
Exam_Name: Theory
Score: 98

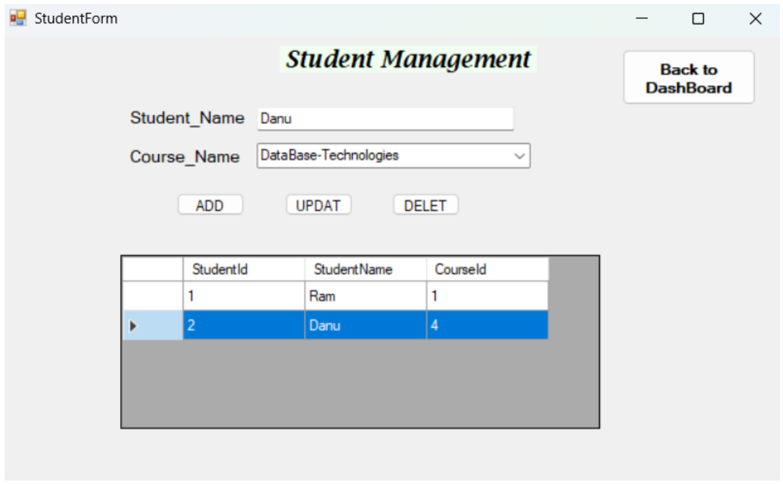
	MarkId	StudentName	SubjectName	ExamN
	1	Ram	C#	Theory
	2	Ram	Html	Practica
▸	3	Danu	C#	Theory
	4	Danu	Python	Viva



TimeTable Management

Subject_Name: Java
Time-Slot: 1.30 PM - 3.30 PM
Room_Name: Lap2

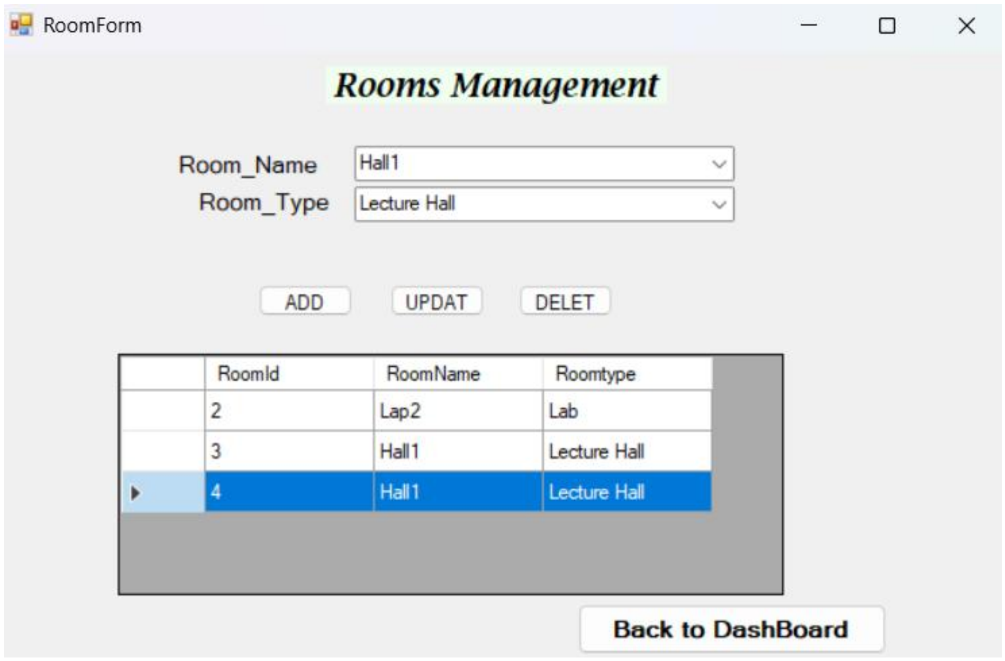
	TimetableId	SubjectName	TimeSlot	RoomN



Student Management

Student_Name: Danu
Course_Name: DataBase-Technologies

	StudentId	StudentName	CourseId
	1	Ram	1
▸	2	Danu	4



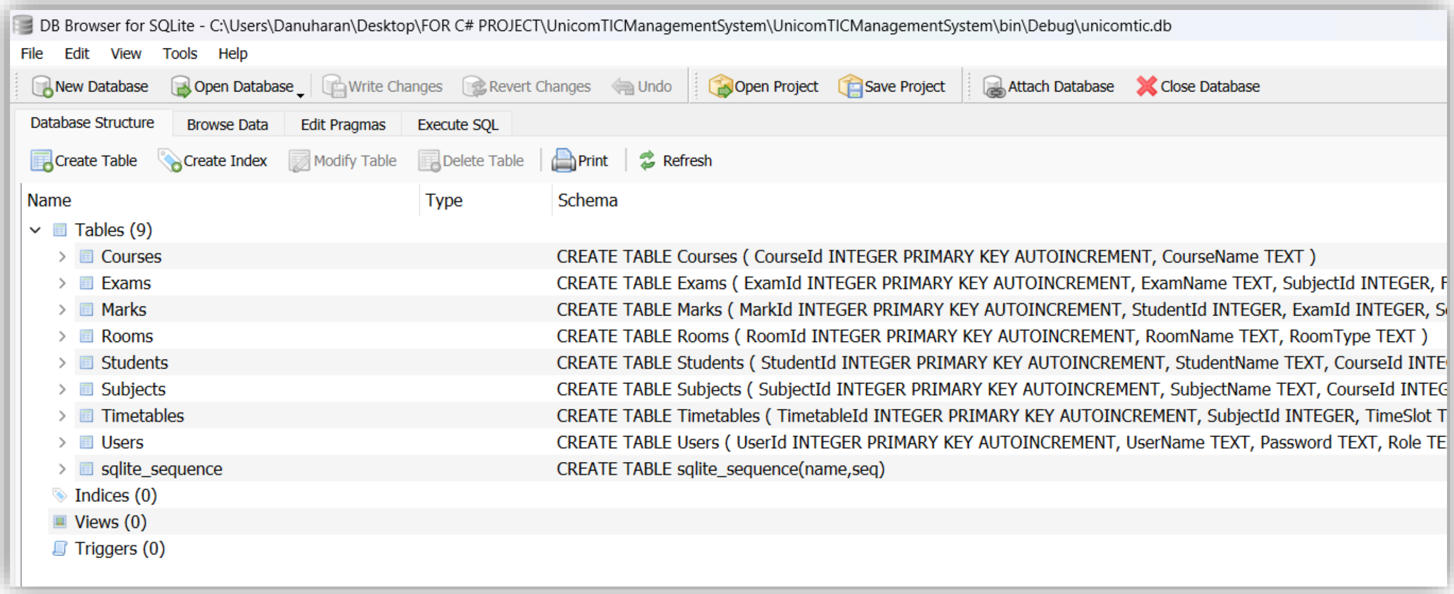
Rooms Management

Room_Name: Hall1
Room_Type: Lecture Hall

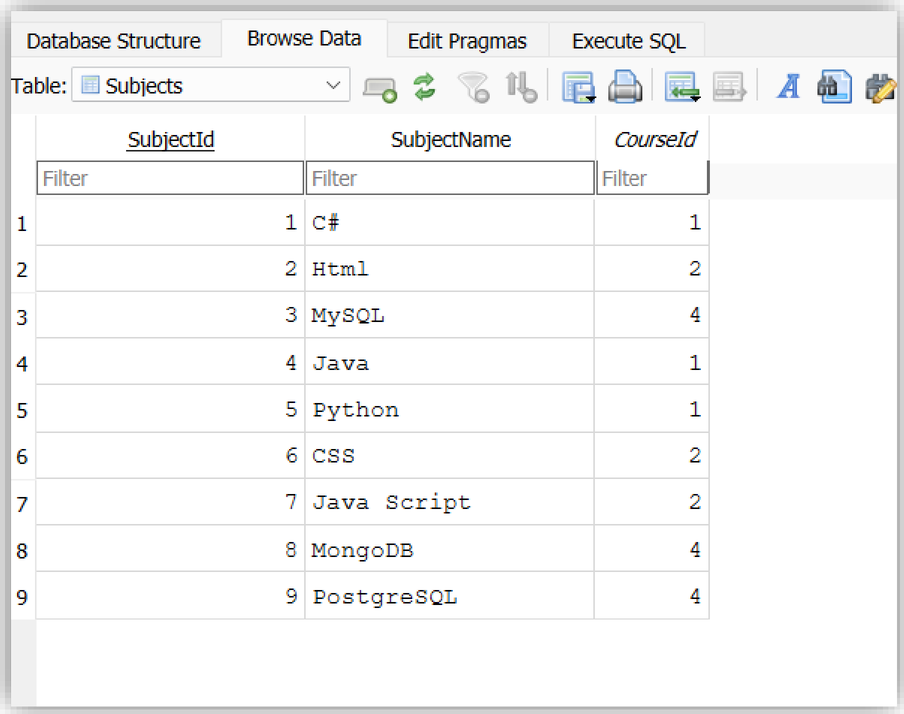
	RoomId	RoomName	Roomtype
	2	Lap2	Lab
	3	Hall1	Lecture Hall
▸	4	Hall1	Lecture Hall

Database Management

The system uses **SQLite** as its database engine to store and manage all information and data. SQLite is a lightweight, file-based database that requires minimal setup, making it ideal for desktop applications.


DB Browser for SQLite - C:\Users\Danuharan\Desktop\FOR C# PROJECT\UnicomTICManagementSystem\UnicomTICManagementSystem\bin\Debug\unicomtic.db

Name	Type	Schema
Tables (9)		
Courses		CREATE TABLE Courses (CourseId INTEGER PRIMARY KEY AUTOINCREMENT, CourseName TEXT)
Exams		CREATE TABLE Exams (ExamId INTEGER PRIMARY KEY AUTOINCREMENT, ExamName TEXT, SubjectId INTEGER, F
Marks		CREATE TABLE Marks (MarkId INTEGER PRIMARY KEY AUTOINCREMENT, StudentId INTEGER, ExamId INTEGER, S
Rooms		CREATE TABLE Rooms (RoomId INTEGER PRIMARY KEY AUTOINCREMENT, RoomName TEXT, RoomType TEXT)
Students		CREATE TABLE Students (StudentId INTEGER PRIMARY KEY AUTOINCREMENT, StudentName TEXT, CourseId INTE
Subjects		CREATE TABLE Subjects (SubjectId INTEGER PRIMARY KEY AUTOINCREMENT, SubjectName TEXT, CourseId INTEG
Timetables		CREATE TABLE Timetables (TimetableId INTEGER PRIMARY KEY AUTOINCREMENT, SubjectId INTEGER, TimeSlot T
Users		CREATE TABLE Users (UserId INTEGER PRIMARY KEY AUTOINCREMENT, UserName TEXT, Password TEXT, Role TE
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
Indices (0)		
Views (0)		
Triggers (0)		



	SubjectId	SubjectName	CourseId
	Filter	Filter	Filter
1	1	C#	1
2	2	Html	2
3	3	MySQL	4
4	4	Java	1
5	5	Python	1
6	6	CSS	2
7	7	Java Script	2
8	8	MongoDB	4
9	9	PostgreSQL	4

OOP Concepts Used

The system applies key Object-Oriented Programming (OOP) principles for better structure, security, and reusability:

- **Class and Object**: The project uses classes for handling users, forms, and database operations. Each form or module (e.g., LoginForm, StudentForm) is treated as an object.
- **Encapsulation**:
  The system separates logic using **Model-Controller** structure. Database queries are handled through dedicated controller classes or methods, preventing direct access to SQL from UI forms. This helps improve security and reduces the risk of **SQL injection attacks**, as parameterized queries are used instead of raw SQL strings.
- **Inheritance**:
  A **base (parent) form** is created to define common properties and methods (e.g., window styling, navigation buttons). Other forms (e.g., StudentForm, AdminForm) inherit from this base form, promoting code reuse and consistent behavior.
- **Polymorphism**
  Methods like Show() or overridden event handlers may work differently in child forms, demonstrating runtime polymorphism.

Future Enhancements

- Add **online student portal** access through a web or mobile version.
- Include features like **attendance tracking**, **exam results**, and **notifications**.
- Improve **UI/UX** with modern design frameworks.
- Add **backup and recovery** features for better data protection.
- Enable **real-time updates** using cloud database integration.

Conclusion

The Unicomtic Management System successfully demonstrates a simple yet functional approach to managing all academic information. By applying Object-Oriented Programming and secure database handling through SQLite, the system ensures modular design, better data management, and ease of use. It lays a strong foundation for further development and feature expansion in the future.