

DATABASE MANAGEMENT SYSTEM

UE21CS351A

MINIPROJECT

CAPSTONE MANAGEMENT SYSTEM

Suhas Gowda Harish - PES1UG21CS633

Trisha Songra - PES1UG21CS677

USER REQUIREMENT SPECIFICATION

The Capstone Management System is a comprehensive software solution designed to facilitate the planning, tracking, assessment, and administration of capstone projects within an educational institution. Capstone projects often serve as a culminating experience for students, where they apply their knowledge and skills to solve real-world problems or complete a substantial project.

In this database we keep track of the students' and mentors' personal as well as academic profiles, maintenance of capstone teams' details including constraints on team size, CGPA and mentor allocation and approval of teams. The projects can be efficiently monitored and assessed as we keep track of their progress status. The panel of judges for the review are meticulously maintained to make sure there are no discrepancies during assessments. The judges present in the specific panel review the teams' projects and provide insightful feedback on a scheduled day and location. Each review is recorded and stored for documentation purposes.

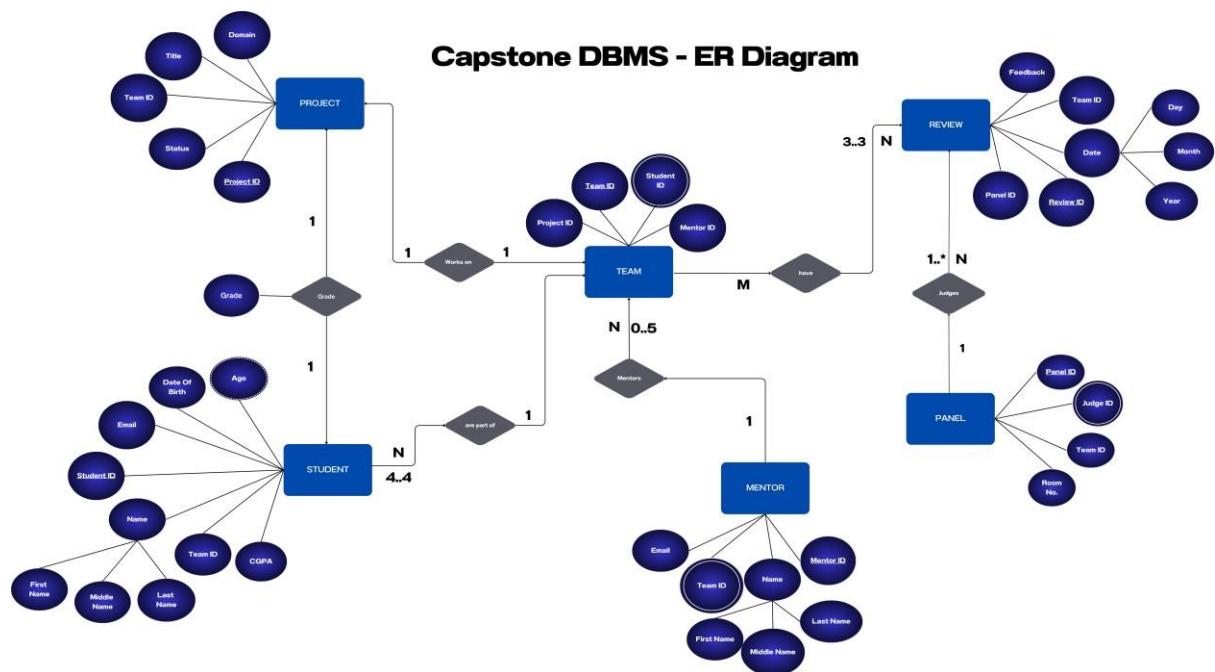
There are many ways in which this database management system helps the various users associated with it, some of them are:

- Efficiently plan, assign, and manage capstone projects, reducing administrative overhead.
- Centralized repository for project documentation, proposals, and progress tracking.
- Reduces manual administrative tasks and paperwork associated with capstone project management.
- Frees up faculty and staff time for more valuable interactions with students.
- Simplifies the creation and management of review panels, including panel member assignments and scheduling.
- Ensures a systematic and organized review process

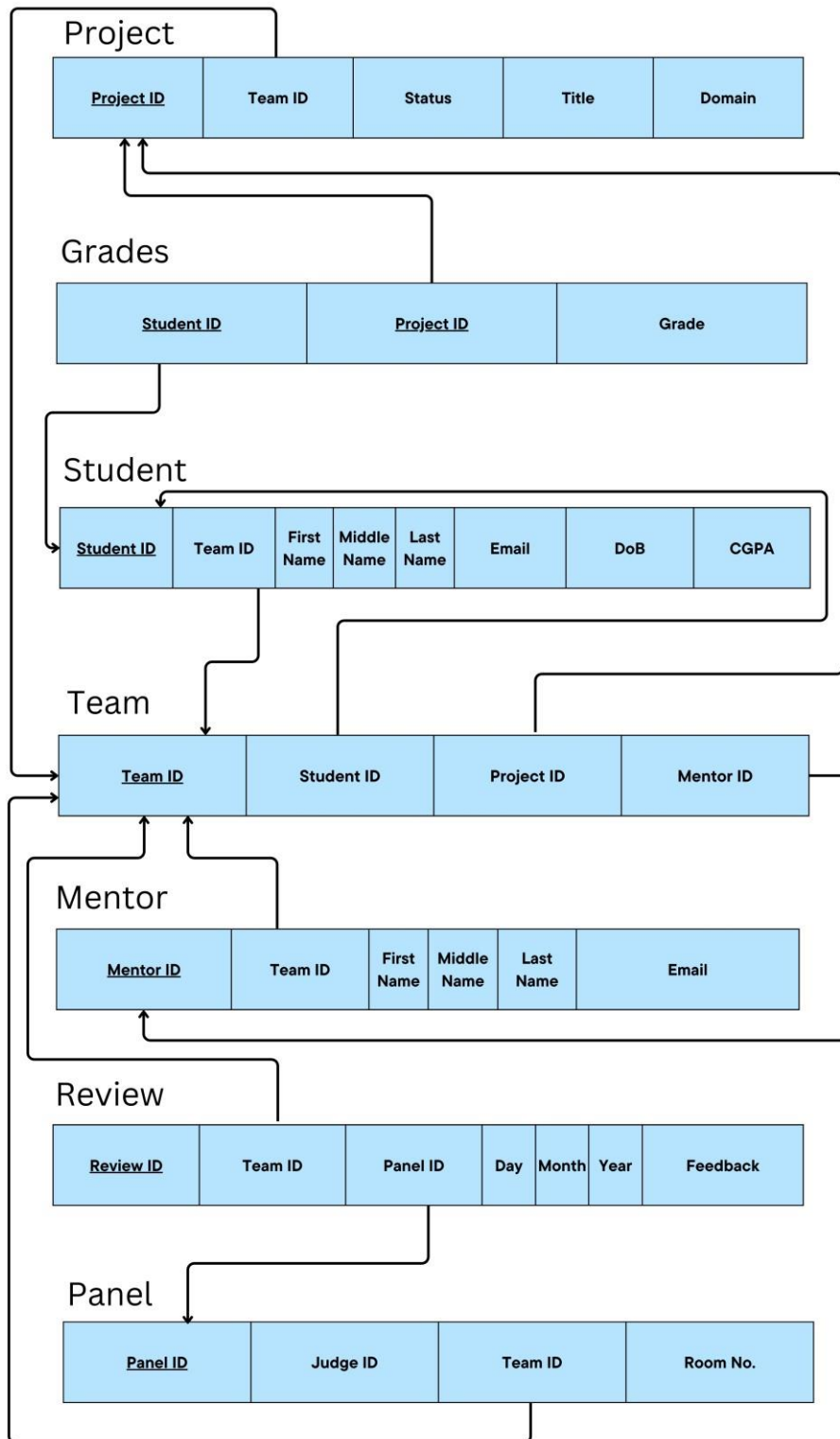
The system responds quickly to user actions, even during peak usage periods. It handles a large number of concurrent users and projects. Data is stored securely and integrity is maintained, user authentication and authorization is robust. The user interface is intuitive and user-friendly, the system is scalable and can accommodate the growth of capstone programs. The system is readily available and reliable, with minimal downtime. Regular data backups and disaster recovery procedures are in place. The system is compatible on various web browsers and mobile devices. It will support multiple operating systems. The system will prevent data duplication and inconsistencies are avoided. Procedures for handling system updates, bug fixes, and feature enhancements should be in place.

Future scope of this project involves:

- Notification features for important dates, meetings, and project updates such as emails or alerts being broadcasted to all the entities of the database
- Additional data security can be provided through user authentication, role-based access control (user classes such as administrative view, student view and the mentor view along with additional access for feedback and grade submission for faculty who are part of the review panel) and data encryption.
- Provide access to students to monitor their project's status and receive timely feedback.



Capstone DBMS - Relational Schema



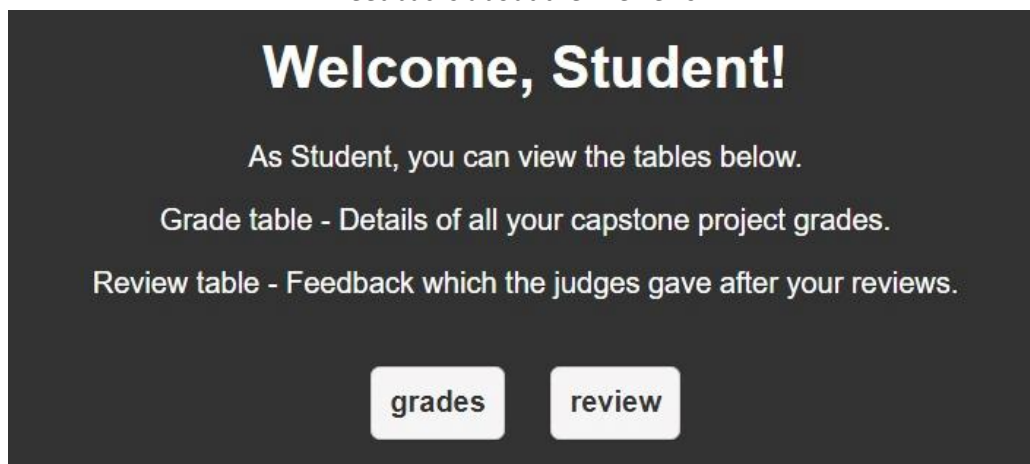
List of Functionalities

1. Login page with credentials for secure access along with error handling for invalid logins.
---put the invalid credentials error also



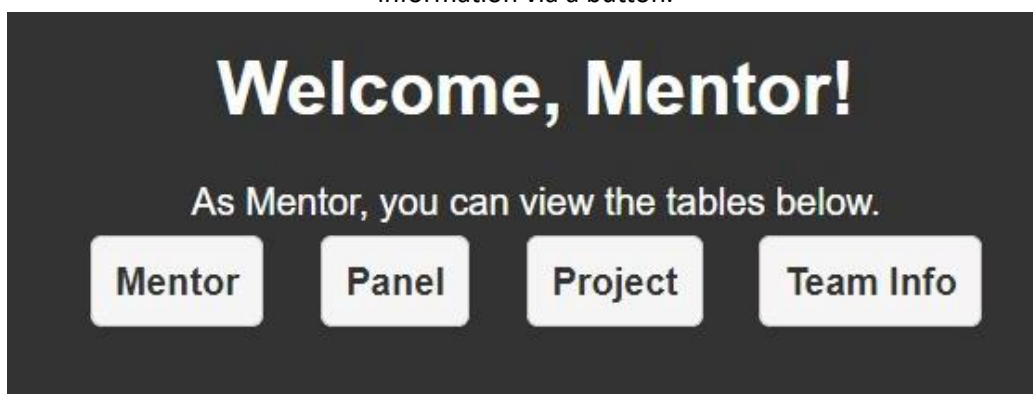
A login form titled "Login To Your Respective Role" on a dark background. It features two white input fields: "Username" and "Password", followed by a "Login" button.

2. There are 3 different views for different roles with different levels of database access/permissions, namely:
Student - Only has view access for getting information about their project grades and panel feedbacks about their reviews.



A welcome screen for a student on a dark background. It says "Welcome, Student!" in large white text. Below it, in smaller white text, it says "As Student, you can view the tables below." followed by two lines: "Grade table - Details of all your capstone project grades." and "Review table - Feedback which the judges gave after your reviews." At the bottom, there are two white buttons labeled "grades" and "review".

- Mentor** – Only has view access to mentor, panel and project tables and can get detailed team information via a button.



A welcome screen for a mentor on a dark background. It says "Welcome, Mentor!" in large white text. Below it, in smaller white text, it says "As Mentor, you can view the tables below." At the bottom, there are four white buttons labeled "Mentor", "Panel", "Project", and "Team Info".

Administrator – Has complete editing (CRUD operation) access to all tables of the schema as well as team information, progress visualisation and team approval access.

Welcome, Admin!

As Admin you have access to all the tables mentioned below, with editing permissions

grades

mentor

mentorteam

panel

paneljudge

project

review

student

team

teamstudent

Team Info

Show Charts

Approve Teams

3. The Admin view has complete access on all tables- viewing the record, inserting a record, updating the record(by checking the checkbox to update only those values) and deleting the record.

Insert Data

PanelID:

JudgeID:

TeamID:

RoomNumber:

Insert Data

Update Data

Select record to update: 1 - 201 - 1 - 111 ▾

PanelID: ☐

JudgeID: ☐

TeamID: ☐

RoomNumber: ☐

Update Data

Delete Data

☐ 201 - 1 - 111

☐ 202 - 2 - 222

☐ 203 - 3 - 333

☐ 204 - 4 - 444

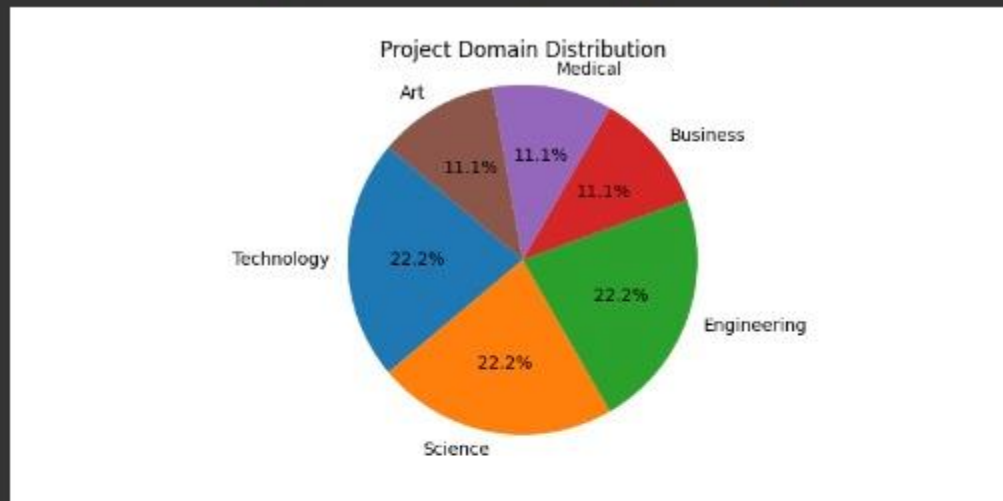
☐ 205 - 5 - 555

Delete Selected

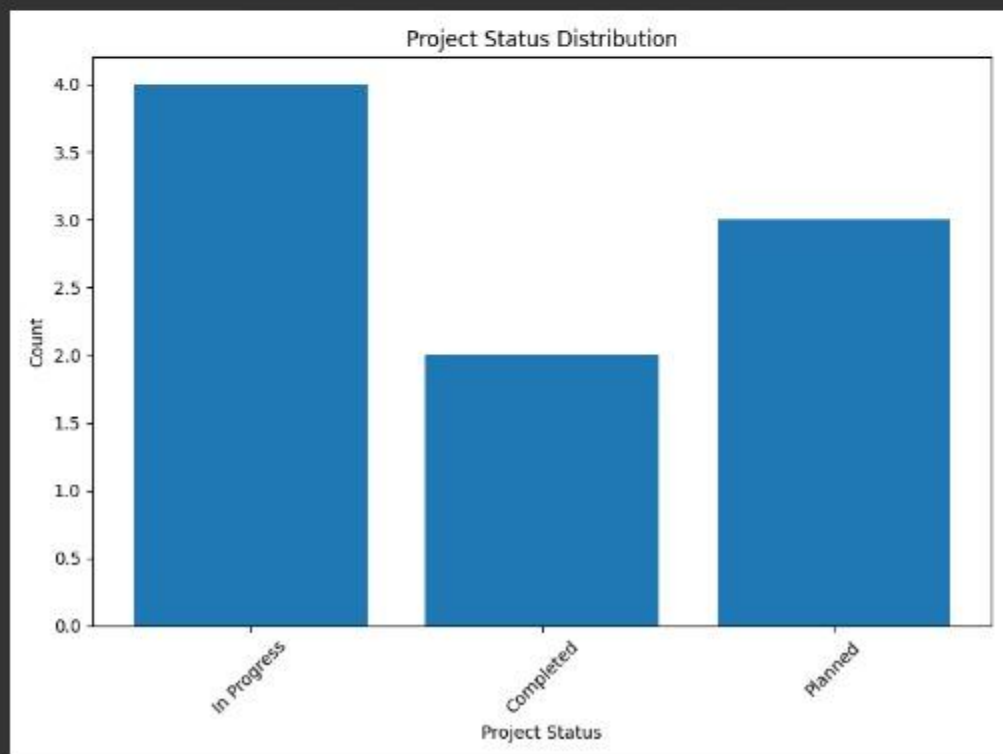
4. Exclusively, the admin can also get a pie-chart visualisation of which domain each project belongs to, a bar chart of the extent of all the teams' progress.

Show Charts

Project Domain Distribution (Pie Chart)



Project Status Distribution (Bar Chart)



5. Admin also has exclusive and specific access to approve, reject, or hold a mentor's team.

Approve Teams					
Mentor ID	Mentor Name	Team ID	Project Title	Domain	Status
201	Mentor1 A. Smith	1	Project A	Technology	Current Status: Pending Change Status: <input type="button" value="Approved"/>
202	Mentor2 B. Johnson	2	Project B	Science	Current Status: Not Approved Change Status: <input type="button" value="Not Approved"/>
203	Mentor3 C. Williams	3	Project C	Engineering	Current Status: Pending Change Status: <input type="button" value="Pending"/>
204	Mentor4 D. Brown	4	Project D	Business	Current Status: Not Approved Change Status: <input type="button" value="Not Approved"/>

6. The insertion operation on Student table is restricted by a constraint on team size i.e. if a team already has 4 members, a 5th cannot be added (error is displayed)

Before Insert:

21	6	Aiden	U.	Hill	aiden.hill@example.com	2002-08-08	7.10
22	6	Grace	V.	Murphy	grace.murphy@example.com	2001-01-19	8.60
23	6	Elijah	W.	Cooper	elijah.cooper@example.com	2000-06-14	9.45
24	6	Suhas Gowda	Gowda	Harish	suhasgowda0606@gmail.com	2003-06-06	9.22

Insert Data

StudentID:

TeamID:

FirstName:

MiddleName:

LastName:

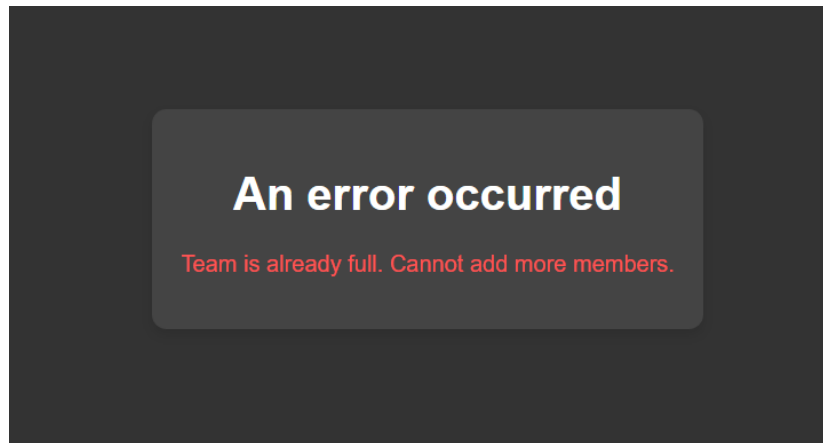
Email:

DOB:

CGPA:

Insert Data

After Insert:



7. The mentors and admins are provided with a button which leads to a drop down list that allows them to see details of each and every team. This has an inbuilt CGPA checker which makes sure that the team is not violating the CGPA variety constraint.

Select Team ID:1Fetch Team Information

Team Information:

Team ID: 1
Project Title: Project A
Project Domain: Technology
Project Status: In Progress
Mentor Name: Mentor1 Smith
Mentor Email: mentor1@example.com
Student Names: John Doe,Jane Smith,Robert Johnson,Emily Wilson
Student Grades: A,B,C,B
Team Status: Pending

CGPA Checker

CGPA Constraint Violated for the selected team.

All Tables

grades Table

StudentID	ProjectID	Grade
1	101	A
2	101	B
3	101	C
4	101	B

Insert Data

StudentID:

ProjectID:

Grade:

Insert Data

mentor Table

MentorID	TeamID	FirstName	MiddleName	LastName	Email	status
201	1	Mentor1	A.	Smith	mentor1@example.com	Pending
202	2	Mentor2	B.	Johnson	mentor2@example.com	Not Approved
203	3	Mentor3	C.	Williams	mentor3@example.com	Pending
204	4	Mentor4	D.	Brown	mentor4@example.com	Not Approved
205	5	Mentor5	E.	Davis	mentor5@example.com	Approved
206	6	Mentor6	F.	Anderson	mentor6@example.com	Not Approved
207	7	Mentor7	G.	Martinez	mentor7@example.com	Approved
208	8	Mentor8	H.	Robinson	mentor8@example.com	Not Approved
209	9	Mentor9	I.	Clark	mentor9@example.com	Approved
210	10	Mentor10	J.	Rodriguez	mentor10@example.com	Not Approved

panel Table

PanelID	JudgeID	TeamID	RoomNumber
1	201	1	111
2	202	2	222
3	203	3	333
4	204	4	444
5	205	5	555

project Table

ProjectID	TeamID	Status	Title	Domain
101	1	In Progress	Project A	Technology
102	2	Completed	Project B	Science
103	3	In Progress	Project C	Engineering
104	4	Planned	Project D	Business
105	5	Planned	Project E	Medical
106	6	Completed	Project F	Art
107	7	In Progress	Project G	Technology
108	8	Planned	Project H	Science
109	9	In Progress	Project I	Engineering
110	10	Planned	Project J	Business

student Table

StudentID	TeamID	FirstName	MiddleName	LastName	Email	DOB	CGPA
1	1	John	A.	Doe	john.doe@example.com	2000-05-15	5.75
2	1	Jane	B.	Smith	jane.smith@example.com	2001-02-20	6.90
3	1	Robert	C.	Johnson	robert.johnson@example.com	2000-11-10	7.60
4	1	Emily	D.	Wilson	emily.wilson@example.com	2002-07-05	8.85
5	2	Michael	E.	Brown	michael.brown@example.com	1999-08-30	9.70
6	2	Sarah	F.	Clark	sarah.clark@example.com	2001-04-25	4.95
7	2	David	G.	White	david.white@example.com	2000-09-12	5.55
8	2	Olivia	H.	Adams	olivia.adams@example.com	2002-01-08	6.80
9	3	William	I.	Martin	william.martin@example.com	2001-06-18	3.65
10	3	Sophia	J.	Anderson	sophia.anderson@example.com	2002-03-22	8.75
11	3	Megan	K.	Harris	megan.harris@example.com	2001-08-17	7.70

review Table

ReviewID	TeamID	PanelID	Day	Month	Year	Feedback
1	1	1	17	11	2023	Excellent work, keep it up.
2	1	1	27	11	2023	Not much progress.
3	1	2	17	12	2023	S grade for you.

Normal Form:

The schema is in Third Normal Form (3NF) and also satisfies the Boyce-Codd Normal Form (BCNF) conditions. No multi-valued dependencies exist. The use of triggers and stored procedures to maintain consistency and update related tables ensure data integrity.

The values are Atomic, Primary, No grouping, No partial dependencies, No transitive dependencies, No non-trivial functional dependencies.

Complex queries(Including aggregate, nested and join queries + CRUD operations), Triggers, Functions(stored procedures):

To retrieve the entire team info by joining tables, grouping by team_id , concatenating the names and grades.

```
SELECT
    Team.TeamID,
    Project.Title AS ProjectTitle,
    Project.Domain AS ProjectDomain,
    Project.Status AS ProjectStatus,
    Mentor.FirstName AS MentorFirstName,
    Mentor.LastName AS MentorLastName,
    Mentor.Email AS MentorEmail,
    GROUP_CONCAT(CONCAT(Student.FirstName, ' ', Student.LastName) ORDER BY Student.StudentID ASC) AS StudentNames,
    GROUP_CONCAT(Grades.Grade ORDER BY Student.StudentID ASC) AS StudentGrades
FROM Team
INNER JOIN Project ON Team.ProjectID = Project.ProjectID
INNER JOIN Mentor ON Team.MentorID = Mentor.MentorID
LEFT JOIN TeamStudent ON Team.TeamID = TeamStudent.TeamID
LEFT JOIN Student ON TeamStudent.StudentID = Student.StudentID
LEFT JOIN Grades ON TeamStudent.StudentID = Grades.StudentID AND Project.ProjectID = Grades.ProjectID
WHERE Team.TeamID = 1
GROUP BY Team.TeamID;
```

Stored procedure to update team table after insertion in mentor and project table.

```
DELIMITER //

-- Create a stored procedure to update Team based on Project data
CREATE PROCEDURE UpdateTeamProjectID()
BEGIN
    -- Declare variables to hold ProjectID and TeamID
    DECLARE project_id INT;
    DECLARE team_id INT;

    -- Declare a variable to check if a row was found
    DECLARE no_more_rows INT DEFAULT 0;

    -- Declare a cursor for selecting Project data
    DECLARE project_cursor CURSOR FOR
        SELECT Project.ProjectID, Team.TeamID
        FROM Project
        JOIN Team ON Project.TeamID = Team.TeamID;

    -- Declare a continue handler to exit the loop when no more rows are found
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET no_more_rows = 1;

    -- Open the cursor
    OPEN project_cursor;

    -- Start processing rows
    read_loop: LOOP
        FETCH project_cursor INTO project_id, team_id;

        -- If no more rows are found, exit the loop
        IF no_more_rows = 1 THEN
```

```

        LEAVE read_loop;
    END IF;

    -- Update the Team table with ProjectID
    UPDATE Team
    SET ProjectID = project_id
    WHERE TeamID = team_id;
END LOOP;

-- Close the cursor
CLOSE project_cursor;
END;
//
DELIMITER ;

CALL UpdateTeamProjectID();

DELIMITER //

-- Create a stored procedure to update Team based on Mentor data
CREATE PROCEDURE UpdateTeamMentorID()
BEGIN
    -- Declare variables to hold MentorID and TeamID
    DECLARE mentor_id INT;
    DECLARE team_id INT;

    -- Declare a variable to check if a row was found
    DECLARE no_more_rows INT DEFAULT 0;

    -- Declare a cursor for selecting Mentor data
    DECLARE mentor_cursor CURSOR FOR
        SELECT Mentor.MentorID, Team.TeamID
        FROM Mentor
        JOIN Team ON Mentor.TeamID = Team.TeamID;

    -- Declare a continue handler to exit the loop when no more rows are found
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET no_more_rows = 1;

    -- Open the cursor
    OPEN mentor_cursor;

    -- Start processing rows
    read_loop: LOOP
        FETCH mentor_cursor INTO mentor_id, team_id;

        -- If no more rows are found, exit the loop
        IF no_more_rows = 1 THEN
            LEAVE read_loop;

```

```

-- Update the Team table with MentorID
UPDATE Team
SET MentorID = mentor_id
WHERE TeamID = team_id;
END LOOP;

-- Close the cursor
CLOSE mentor_cursor;
END;
//
DELIMITER ;

CALL UpdateTeamMentorID();

```

Automatically filling the team_student table

```

CREATE TRIGGER populate_team_student
AFTER INSERT ON Student
FOR EACH ROW
BEGIN
    INSERT INTO TeamStudent (TeamID, StudentID)
    VALUES (NEW.TeamID, NEW.StudentID);
END;
//
DELIMITER ;

```

Automatically filling the mentor_team table

```

DELIMITER //

CREATE TRIGGER populate_mentor_team
AFTER INSERT ON Mentor
FOR EACH ROW
BEGIN
    INSERT INTO MentorTeam (TeamID, MentorID)
    VALUES (NEW.TeamID, NEW.MentorID);
END;
//DELIMITER ;

```

Providing visualisation via charts

```
@app.route('/show_charts')
def show_charts():
    conn = pymysql.connect(host=server,user=username,password=password,database=database)

    cursor = conn.cursor()

    # Query for Domain counts
    cursor.execute("SELECT Domain, COUNT(*) AS DomainCount FROM Project GROUP BY Domain")
    domain_data = cursor.fetchall()
    domain_labels, domain_counts = zip(*domain_data)

    # Query for Status counts
    cursor.execute("SELECT Status, COUNT(*) AS StatusCount FROM Project GROUP BY Status")
    status_data = cursor.fetchall()
    status_labels, status_counts = zip(*status_data)

    cursor.close()
    conn.close()

    # Create a pie chart for Domain counts
    plt.figure(figsize=(8, 4))
    plt.pie(domain_counts, labels=domain_labels, autopct='%1.1f%%', startangle=140)
    plt.axis('equal')
    plt.title('Project Domain Distribution')
    plt.savefig('static/domain_pie_chart.png')
    plt.clf()

    # Create a bar chart for Status counts
    plt.figure(figsize=(8, 6))
    plt.bar(status_labels, status_counts)
    plt.xlabel('Project Status')
    plt.ylabel('Count')
    plt.title('Project Status Distribution')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.savefig('static/status_bar_chart.png')
    plt.clf()

    return render_template('show_charts.html')
```

Granting permissions to different users

```
-- Create a user for the Student role with limited access
CREATE USER 'student'@'localhost';

-- Grant SELECT privileges on specific tables for Student
GRANT SELECT ON Universitydb.Grade TO 'student'@'localhost';
GRANT SELECT ON Universitydb.Review TO 'student'@'localhost';

-- Create a user for the Mentor role
CREATE USER 'mentor'@'localhost';

-- Grant various privileges to the Mentor user for specified tables
GRANT SELECT, INSERT, UPDATE ON Universitydb.Mentor TO 'mentor'@'localhost';
GRANT SELECT, INSERT, UPDATE ON Universitydb.Panel TO 'mentor'@'localhost';
GRANT SELECT, INSERT, UPDATE ON Universitydb.Project TO 'mentor'@'localhost';
GRANT SELECT, INSERT, UPDATE ON Universitydb.Team TO 'mentor'@'localhost';

-- Create a user for the Admin role
CREATE USER 'admin'@'localhost';

-- Grant full access to all tables for the Admin user
GRANT ALL PRIVILEGES ON Universitydb.* TO 'admin'@'localhost';
```