A Report of the Mini Project

Lab Internal Marks Management System

Roll No. : 2023103533

Name : Kishore K

Semester : IV

Degree: Bachelor of Engineering (B.E.)

Branch: Computer Science and Engineering (CSE)

Batch: 2023-2027

For : CS23401- Database Management Systems

Under the Guidance of: Dr. D. Shiloah Elizabeth - Anna University

Date of Submission: 28.04.2025

CRICKET RATING MANAGEMENT SYSTEM

Objective:

The **Cricket Rating Management System (CRMS)** is a database-driven application designed to efficiently track and manage cricket player ratings based on their performances in various matches. The system provides structured data storage for players, teams, matches, and performance statistics while ensuring consistency, security, and real-time analytics.

The CRMS enables **team managers, analysts or administrators** to record match details, player statistics, and rankings. It calculates ratings based on predefined performance metrics such as runs, wickets, strike rates, and averages. The system also maintains records of umpires, stadiums, weather conditions, and match schedules to offer a **comprehensive view of the cricket ecosystem**.

To ensure **data integrity**, the system follows **Boyce-Codd Normal Form (BCNF) normalization** to remove redundancy and improve efficiency. It enforces **relational constraints** through primary and foreign keys, optimizing query performance. Additionally, role-based access control (RBAC) secures sensitive data while enabling authorized users to modify or retrieve information.

Description of the mini world:

The CRMS is designed for **real-time query execution**, allowing users to fetch top-rated players, analyze team performances. Future enhancements may include **AI-driven predictive analytics, integration with live match feeds, and automated player performance forecasting**.

By providing a **structured, efficient, and scalable** solution for cricket data management, the CRMS helps cricket analysts, teams, and enthusiasts make informed decisions based on real-time player ratings and match insights.

Team Management

- Create, update, and manage cricket teams.
- Assign players to teams and manage team compositions.

Match Management

- Schedule matches between teams.
- Input match results and update individual player and team statistics.

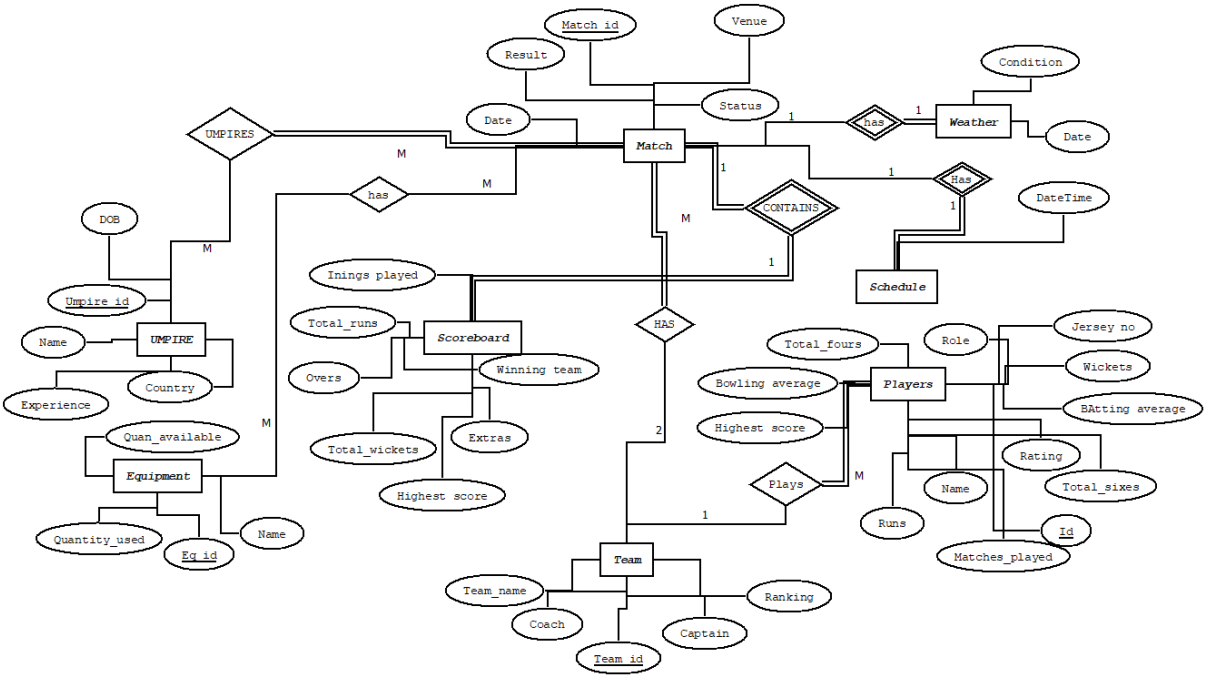
Rating Calculation

- Auto-calculate player and team ratings based on performance metrics (e.g., batting average, bowling economy, fielding impact).

Leaderboard System

- Display top players and teams based on current ratings.

ER Diagram:



Final Relational Schema developed:

1. Players

Column Name	Data Type	Constraints
Player_ID	INT	PRIMARY KEY
Name	VARCHAR(100)	
Jersey_Number	INT	UNIQUE
Role	VARCHAR(50)	
Team_ID	INT	FOREIGN KEY → Teams(Team_ID)

2. Player_Statistics

Column Name	Data Type	Constraints
Stat_ID	INT	PRIMARY KEY
Player_ID	INT	FOREIGN KEY → Players(Player_ID)
Matches_Played	INT	
Runs	INT	
Wickets	INT	
Total_Sixes	INT	
Total_Fours	INT	
Highest_Score	INT	
Strike_Rate	FLOAT	
Batting_Average	FLOAT	
Bowling_Average	FLOAT	
Rating	FLOAT	

---

3. Teams

Column Name	Data Type	Constraints
Team_ID	INT	PRIMARY KEY
Team_Name	VARCHAR(100)	
Country	VARCHAR(100)	

4. Team_Coach_Captain

Column Name	Data Type	Constraints
Team_ID	INT	FOREIGN KEY → Teams(Team_ID)
Coach	VARCHAR(100)	
Captain_ID	INT	UNIQUE, FOREIGN KEY → Players(Player_ID)
Ranking	INT	
PRIMARY KEY	(Team_ID, Captain_ID)	

---

5. Matches

Column Name	Data Type	Constraints
Match_ID	INT	PRIMARY KEY
Date	DATE	
Venue	VARCHAR(255)	
Team1_ID	INT	FOREIGN KEY → Teams(Team_ID)

Column Name	Data Type	Constraints
Team2_ID	INT	FOREIGN KEY → Teams(Team_ID)
Match_Status	VARCHAR(50)	
Result	VARCHAR(255)	

---

6. Match_Umpires

Column Name	Data Type	Constraints
Match_ID	INT	FOREIGN KEY → Matches(Match_ID)
Umpire_ID	INT	FOREIGN KEY → Umpires(Umpire_ID)
PRIMARY KEY	(Match_ID, Umpire_ID)	

---

7. Scoreboard

Column Name	Data Type	Constraints
Match_ID	INT	PRIMARY KEY, FOREIGN KEY → Matches(Match_ID)
Innings_Played	INT	
Total_Runs	INT	
Total_Wickets	INT	
Overs	FLOAT	
Extras	INT	
Highest_Score	INT	
Winning_Team_ID	INT	FOREIGN KEY → Teams(Team_ID)

8.Umpires

Column Name	Data Type	Constraints
Umpire_ID	INT	PRIMARY KEY
Name	VARCHAR(100)	
Country	VARCHAR(100)	
Experience	INT	
Date_of_Birth	DATE	

---

9. Weather

Column Name	Data Type	Constraints
Match_ID	INT	PRIMARY KEY, FOREIGN KEY → Matches(Match_ID)
Forecast_Date	DATE	
Condition	VARCHAR(50)	

---

10. Equipment

Column Name	Data Type	Constraints
Equipment_ID	INT	PRIMARY KEY
Name	VARCHAR(100)	

---

11. Match_Equipment

Column Name	Data Type	Constraints
Match_ID	INT	FOREIGN KEY → Matches(Match_ID)
Equipment_ID	INT	FOREIGN KEY → Equipment(Equipment_ID)
Quantity_Used	INT	

Column Name	Data Type	Constraints
PRIMARY KEY	(Match_ID, Equipment_ID)	

---

12. Schedule

Column Name	Data Type	Constraints
Match_ID	INT	UNIQUE, FOREIGN KEY → Matches(Match_ID)
Match_DateTime	DATETIME	

Tools/ Softwares used:

 MySQL
→ To store my cricket data (players, teams, matches, scores, etc.)
→ Tool: Install **MySQL Server** and maybe use **MySQL Workbench** to easily manage my database.

 Node.js (JavaScript runtime)
→ Built my server API to connect React with MySQL.

 Express.js (on top of Node.js)
→ A fast Node.js framework for handling routes (API endpoints).

 MySQL Node Package
→ mysql2 package to connect Node.js to MySQL.

Functionality: **Update Player Statistics (and Auto-Update Rating):**

Database Concept Used:

✅ **Update Operation** (UPDATE query)
✅ **Triggers** (MySQL AFTER UPDATE Trigger)
✅ **Automatic Calculation** (Rating auto-recalculated inside trigger)

- When **player statistics** are updated (Matches Played, Runs, etc.), a **MySQL trigger** automatically recalculates the player's **Rating**.
- This ensures **no manual recalculation** is needed from Node.js or React — the database maintains integrity itself.
- **Code:**

(1) React Frontend Code (Submitting updated player data)

```
const handleUpdateSubmit = async (e) => {

e.preventDefault();

try {

await axios.put(`http://localhost:5000/api/admin/players/${updateData.Player_ID}`, updateData);

setShowUpdateModal(false);

fetchPlayers();

} catch (err) {

console.error('Error updating player:', err);

}

};
```

(2) Node.js Backend Code (API to update player stats)

```
router.put('/players/:id', async (req, res) => {

const playerId = req.params.id;

const {

Matches_Played,

Runs,

Wickets,

Total_Sixes,

Total_Fours,

Highest_Score,

Batting_Average,
```

```
Bowling_Average

} = req.body;


try {

const [existingPlayer] = await db.query('SELECT * FROM players WHERE
Player_ID = ?', [playerId]);

if (existingPlayer.length === 0) {

return res.status(404).json({ error: 'Player not found' });

}


await db.query(

`UPDATE player_statistics

SET Matches_Played = ?, Runs = ?, Wickets = ?,

Total_Sixes = ?, Total_Fours = ?, Highest_Score = ?, Batting_Average = ?,
Bowling_Average = ?

WHERE Player_ID = ?`,

[

Matches_Played,

Runs,

Wickets,

Total_Sixes,

Total_Fours,

Highest_Score,

Batting_Average,

Bowling_Average,

Number(playerId)

]
```

);

res.status(200).json({ message: 'Player updated successfully' });

} catch (error) {

console.error('Error updating player:', error);

res.status(500).json({ error: 'Failed to update player' });

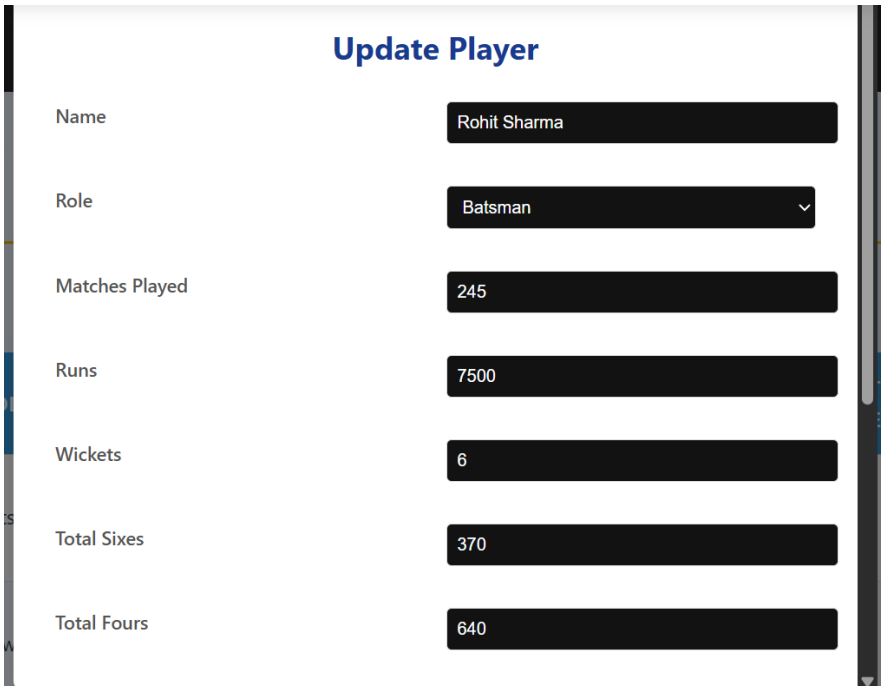
}

});

(3) MySQL Trigger Code (Auto-update player rating)

```
CREATE TRIGGER AfterPlayerStatsUpdate
AFTER UPDATE ON player_statistics
FOR EACH ROW
BEGIN
SET NEW.Rating = CASE
WHEN NEW.Matches_Played > 0 THEN (
(COALESCE(NEW.Runs, 0) * 0.5) +
(COALESCE(NEW.Strike_Rate, 0) * 2) +
(COALESCE(NEW.Total_Sixes, 0) * 2) +
(COALESCE(NEW.Total_Fours, 0) * 1) +
(COALESCE(NEW.Wickets, 0) * 20) -
(COALESCE(NEW.Bowling_Average, 0) * 1.5) +
(COALESCE(NEW.Highest_Score, 0) * 1) +
(COALESCE(NEW.Matches_Played, 0) * 5)
) / COALESCE(NEW.Matches_Played, 1)
ELSE 0
END;
```

END;



Functionality: Register New Match

Stored Procedure (Register_New_Match)

✅ **Multi-table Insertions** (Matches, Schedule, Weather_Condition)

✅ **Error Handling** (MySQL SQLEXCEPTION Handler inside procedure)

- **One button click** in React triggers a **stored procedure** at the database level, which:
  - Inserts the match into Matches table
  - Schedules the match in Schedule table
  - Adds weather condition into Weather_condition table
- If inserting weather fails, error handling inside procedure captures it.

```
const handleSubmit = async (e) => {
  e.preventDefault();
```

```
    console.log(formData);

    const response = await
axios.post('http://localhost:5000/api/admin/register-match', formData);

    alert('Match registered successfully!');

    navigate('/admin');

};Code:
```

- **(1) React Frontend Code (Form Submit)**

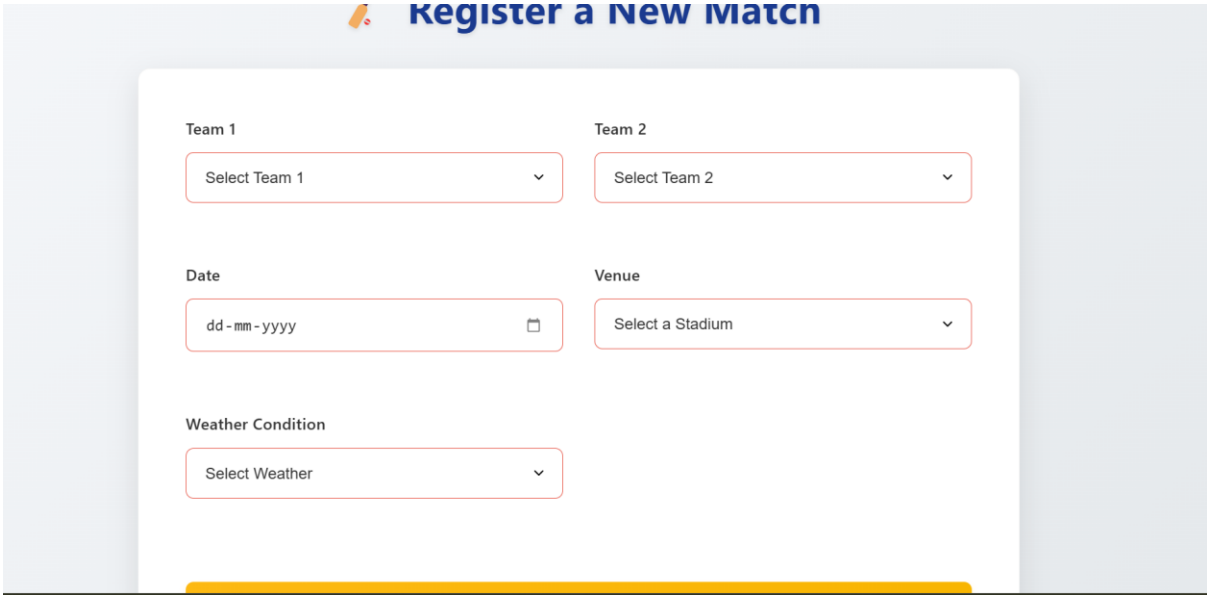
```
router.post('/register-match', async (req, res) => {
  const { date: mdate, venue, team1, team2, time = "19:30:00", weather } =
req.body;
  const matchDateTime = `${mdate} ${time}`;
  const formattedMDate = moment(mdate).format('YYYY-MM-DD');
  const formattedMatchDateTime = moment(matchDateTime).format('YYYY-
MM-DD HH:mm:ss');

  try {
    await db.query(
      'CALL Register_New_Match(?, ?, ?, ?, ?, ?)',
      [formattedMDate, venue, Number(team1), Number(team2),
formattedMatchDateTime, weather]
    );
    res.status(200).json({ message: 'Match registered successfully' });
  } catch (error) {
    console.error('Error registering match:', error);
    res.status(500).json({ message: 'Error registering match' });
  }
});
```

MySQL Stored Procedure (Database Side):

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Register_New_Match`(

    IN p_Date DATE,

    IN p_Venue VARCHAR(255),

    IN p_Team1_ID INT,

    IN p_Team2_ID INT,

    IN p_Match_DateTime DATETIME,

    IN p_Weather_Condition VARCHAR(50)
```

```
)
BEGIN
    DECLARE new_match_id INT;
    DECLARE weather_insert_error INT DEFAULT 0;

    -- Find new match_id
    SELECT IFNULL(MAX(match_id), 0) + 1 INTO new_match_id FROM Matches;

    -- Insert into Matches table
    INSERT INTO Matches (match_id, mdate, Venue, Team1_ID, Team2_ID,
Match_Status, Result)
    VALUES (new_match_id, p_Date, p_Venue, p_Team1_ID, p_Team2_ID,
'Scheduled', NULL);

    -- Insert into Schedule table
    INSERT INTO Schedule (match_id, match_datetime)
    VALUES (new_match_id, p_Match_DateTime);

    -- Try inserting into Weather_condition table
    BEGIN
        DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
        SET weather_insert_error = 1;

        INSERT INTO Weather_condition (match_id, mdate, mcondition)
        VALUES (new_match_id, p_Date, p_Weather_Condition);
```

```
    SET weather_insert_error = 0;
  END;

  -- If weather insert fails, send custom error
  IF weather_insert_error = 1 THEN
    SELECT 'Weather insert failed' AS ErrorMessage;
  END IF;
END;
```



Functionality: Insert Scoreboard:

Database Concept Used:

✅ **Stored Procedure (InsertIntoScoreboardSafe)**
✅ **Uniqueness Enforcement (Match_ID must be unique in scoreboard)**
✅ **Error Signaling using SIGNAL SQLSTATE in MySQL**

- **One button click triggers an insert into the scoreboard table, if the match does not already exist.**
- **If the Match_ID is already present, a custom error is thrown and caught cleanly in backend.**

Code:

- **(1) React Frontend Code (Form Submit)**

```
const handleSubmit = async (e) => {
   e.preventDefault();
   try {
     await axios.post('http://localhost:5000/api/admin/insertScoreBoard', {
       matchId,
       ...formData,
     });
     alert('Scoreboard updated successfully!');
   } catch (error) {
     console.error('Error inserting into scoreboard:', error);
     alert('Failed to insert into scoreboard.');
   }
};
```

(2) Node.js Backend Code (Route to Insert Scoreboard Data)

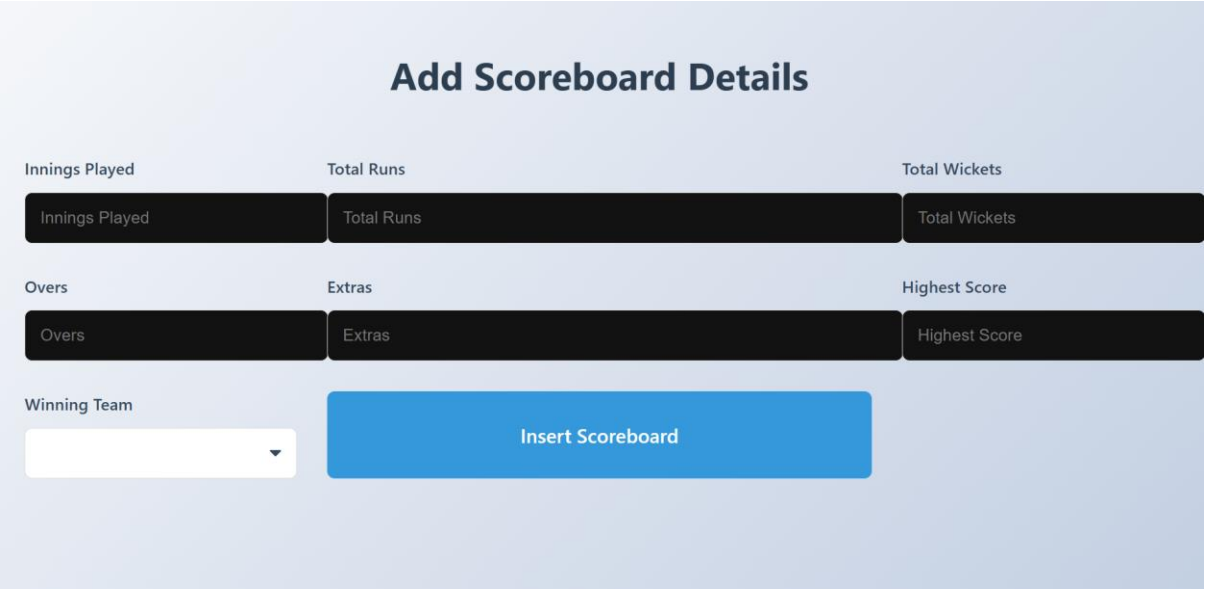
```
router.post('/insertScoreBoard', async (req, res) => {
  const { matchId, inningsPlayed, totalRuns, totalWickets, overs, extras,
highestScore, winningTeamId } = req.body;

  try {
    const [result] = await db.query(
      `CALL InsertIntoScoreboardSafe(?, ?, ?, ?, ?, ?, ?, ?)`,
      [
        matchId,
        inningsPlayed,
        totalRuns,
        totalWickets,
        overs,
        extras,
        highestScore,
        winningTeamId,
      ]
    );

    res.status(200).json({ message: 'Scoreboard inserted successfully.' });
  } catch (error) {
    console.error('Error inserting scoreboard:', error);
```

```
      if (error.code === '45000') {
        return res.status(400).json({ message: 'Match_ID already exists in
    scoreboard!' });
      }

      res.status(500).json({ message: 'Internal Server Error' });
    }
  });
```

(3) MySQL Stored Procedure (Database Side)

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `InsertIntoScoreboardSafe`(

  IN p_Match_ID INT,

  IN p_Innings_Played INT,

  IN p_Total_Runs INT,

  IN p_Total_Wickets INT,

  IN p_Overs FLOAT,

  IN p_Extras INT,

  IN p_Highest_Score INT,

  IN p_Winning_Team_ID INT

)

BEGIN

  -- Check if Match_ID already exists

  IF NOT EXISTS (SELECT 1 FROM scoreboard WHERE Match_ID = p_Match_ID) THEN

    INSERT INTO scoreboard (

      Match_ID,

      Innings_Played,

      Total_Runs,

      Total_Wickets,

      Overs,

      Extras,

      Highest_Score,
```

```
        Winning_Team_ID
    ) VALUES (
        p_Match_ID,
        p_Innings_Played,
        p_Total_Runs,
        p_Total_Wickets,
        p_Overs,
        p_Extras,
        p_Highest_Score,
        p_Winning_Team_ID
    );
  ELSE
    -- Custom error if Match_ID already inserted
    SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: Match_ID already exists in scoreboard!';
  END IF;
END;
```



Functionality: Dynamic Team Leaderboard with Auto-Ranking:

Database Concept Used:

✅ **AFTER INSERT Trigger (update_team_ranking_after_win)**
✅ **Window Functions (RANK() OVER (ORDER BY Wins DESC))**
✅ **Temp Table usage to calculate wins**
✅ **Dynamic Fetch in React frontend using axios.get**

Detailed Flow:

---

(1) Trigger: update_team_ranking_after_win

- **When: After a new record is inserted into scoreboard**
- **Action:**
  - **Count all wins for each team.**
  - **Rank teams dynamically based on wins.**
  - **Update the Ranking field inside Team_Coach_Captain table.**

```
CREATE TRIGGER update_team_ranking_after_win

AFTER INSERT ON scoreboard

FOR EACH ROW

BEGIN

  DROP TEMPORARY TABLE IF EXISTS Temp_Team_Wins;


  CREATE TEMPORARY TABLE Temp_Team_Wins (

    Team_ID INT,

    Wins INT

  );


  INSERT INTO Temp_Team_Wins (Team_ID, Wins)
```

```
    SELECT Winning_Team_ID, COUNT(*) AS Wins
    FROM Scoreboard
    GROUP BY Winning_Team_ID;

    UPDATE Team_Coach_Captain tcc
    JOIN (
        SELECT Team_ID, RANK() OVER (ORDER BY Wins DESC) AS NewRank
        FROM Temp_Team_Wins
    ) ranked_wins
    ON tcc.Team_ID = ranked_wins.Team_ID
    SET tcc.Ranking = ranked_wins.NewRank;

    DROP TEMPORARY TABLE Temp_Team_Wins;
END;
```

(2) API Route to Fetch Leaderboard:

```
router.get("/leaderboard", async (req, res) => {
  const query = `
    SELECT
      Team_ID,
      Team_Name,
      Get_Team_Wins(Team_ID) AS Matches_Won,
      Get_Team_Wins(Team_ID) * 2 AS Points
    FROM Teams
    ORDER BY Points DESC;
  `;
```

```
  try {
    const [rows] = await db.query(query);
    res.json(rows);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

(3) React Frontend (Leaderboard Fetch in useEffect)

```
useEffect(() => {
    axios.get('http://localhost:5000/api/leaderboard')
      .then((res) => setTeams(res.data))
      .catch((err) => console.error('Error fetching leaderboard:', err));
}, []);
```

RANK	TEAM NAME	MATCHES WON	POINTS
1	Chennai Super Kings	2	4
2	Mumbai Indians	0	0
3	Royal Challengers Bangalore	0	0
4	Kolkata Knight Riders	0	0
5	Rajasthan Royals	0	0
6	Sunrisers Hyderabad	0	0
7	Delhi Capitals	0	0
8	Punjab Kings	0	0

Functionality:Umpire Management System

Database Concepts Used:

- **Normal Insert (adding a new umpire)**
- **Normal Delete (removing an umpire)**
- **Primary Key Management (manually calculating next Umpire_ID)**

Backend React Functions:

1) POST /umpires → Add a New Umpire

```
const [rows] = await db.query('SELECT MAX(Umpire_ID) AS maxId FROM Umpires');

const newId = (rows[0].maxId || 0) + 1;
```

2) DELETE /umpires/:id → Delete Umpire by ID

```
await Umpire.delete(id);
```

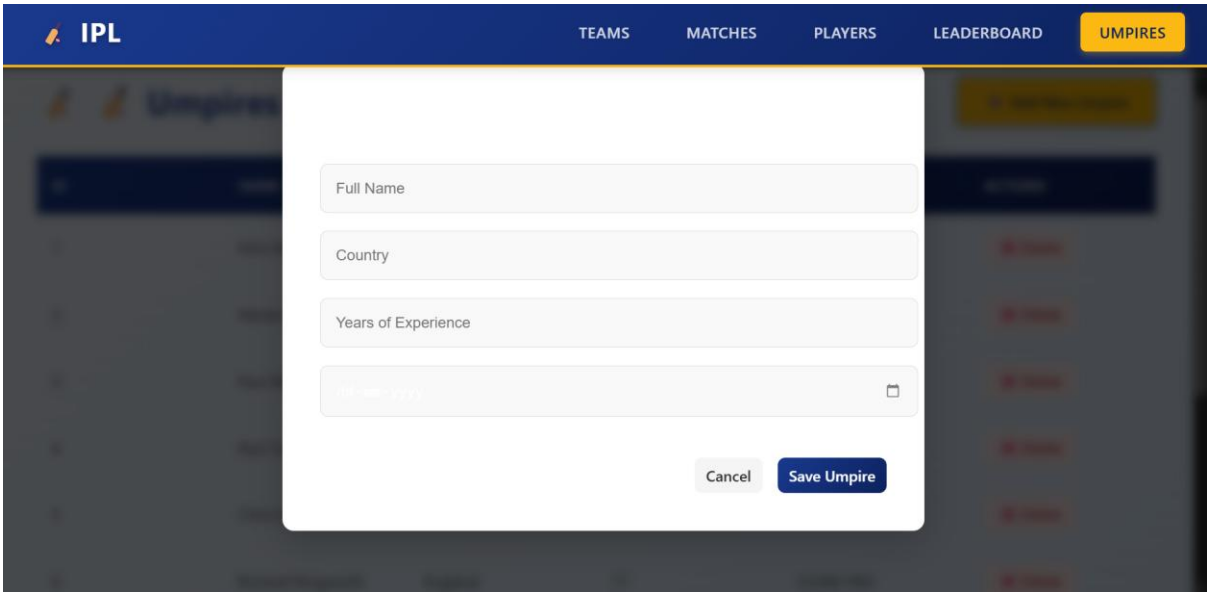
Frontend React Functions:

handleAddUmpire

- **Calls POST API with the form data.**
- **After success: resets form, fetches updated umpire list.**

handleDelete

- **Confirms with the user (window.confirm).**
- **Calls DELETE API to remove the umpire.**
- **Refreshes the umpire list after deletion.**



Conclusion:

In this project, we successfully implemented a cricket management system that handles core functionalities like player updates, match registrations, scoreboard insertions, leaderboard rankings, and umpire management. We used:

- **Normal Insert and Delete operations for adding and removing umpires.**
- **Stored Procedures and Triggers for automating critical tasks such as updating player ratings and recalculating team rankings after each match.**
- **Frontend integration (React + Axios) to allow seamless user interactions.**
- **Backend REST APIs (Node.js + Express.js) connected to a MySQL database for efficient data storage and retrieval.**

Future Scope:

Real-Time Score Updates

Use WebSockets to update live scores instantly without refreshing the page

Role-Based Access Control

Separate permissions for Admins, Coaches, and Captains.

Player and Team Analytics

Expand player and team profiles with detailed performance charts (e.g., match-by-match progression graphs).

References:

MySQL Documentation —
MySQL 8.0 Reference Manual
https://dev.mysql.com/doc/

Express.js Guide —
Express.js Web Framework
https://expressjs.com/

SQL Triggers and Stored Procedures —
TutorialsPoint: SQL - Triggers and Procedures
https://www.tutorialspoint.com/sql/sql-triggers.html