**Database Management Systems Lab**

**Project Phase 2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.No.** | **Name** | **PRN** | **Branch** | **Division** |
| 1 | Josh Dagla | 22070122089 | CS | B1 |
| 2 | Kasibhatta Suprabhat | 22070122095 | CS | B1 |
| 3 | Kshitij Gurbuxani | 22070122097 | CS | B1 |
| 4 | Krish Panchal | 22070122102 | CS | B1 |

# **Title of the project:**

**IPL Stats Management System**

# **About the project:**

Cricket, a globally cherished sport, has undergone substantial evolution, captivating millions of fans, and enthusiasts. In today's era, acquiring comprehensive and precise cricket statistics proves challenging for players, analysts, and enthusiasts alike. Navigating through diverse platforms and records can be time-intensive, often resulting in overwhelming information. To counter these challenges, we advocate for the creation of a Cricket Statistics Application fortified with a robust Database Management System. This application aims to simplify the process of accessing cricket statistics, providing users with an intuitive platform to explore player performances, team statistics, and match analyses effortlessly. The application permits users, whether they are cricket fans, players, or analysts, to peruse an extensive array of cricket statistics, encompassing individual player records, team performances, and match-specific details. Users can delve into player averages, strike rates, and various other key metrics, offering a comprehensive glimpse into a player's cricketing journey. To enhance user engagement, the application offers a guest mode, enabling users to explore basic statistics without the necessity for registration. Nevertheless, for access to more intricate features and personalized experiences, users can register on the platform. Registered users can relish additional functionalities, including the ability to save favorite players and receive personalized match recommendations. In summary, the Cricket Statistics Application envisions itself as a comprehensive platform bridging the divide between cricket enthusiasts and the vast pool of statistical information generated by the sport. This endeavor aims to furnish users with a seamless and enriching experience, providing a centralized hub for cricket statistics, and fostering a deeper connection among fans, players, and the continually evolving world of cricket.

# **Addressed paint points:**

So far, there has not been a dedicated app or website for detailed statistics of the IPL. Although some websites like ESPNcricinfo and Cricbuzz have a few IPL stats in their archives section, there is no application or website available for detailed statistics. In our IPL stats management system, users can find detailed stats of each season that has occurred so far, presented in an organized manner.

Few pain points that we address from existing IPL stats apps and highlighting what is better and new in our app:

**1. Comprehensive Data Model:**

Pain Point: Existing apps might lack a comprehensive data model, leading to fragmented or incomplete stats.

Improvement: Our app integrates multiple tables covering team ownership, player stats (batting and bowling), player information, IPL details (teams, matches, seasons), standings, and stadiums, providing a holistic view of IPL statistics.

**2. Player-Centric Stats:**

Pain Point: Some apps may focus more on team statistics, overlooking detailed player stats.

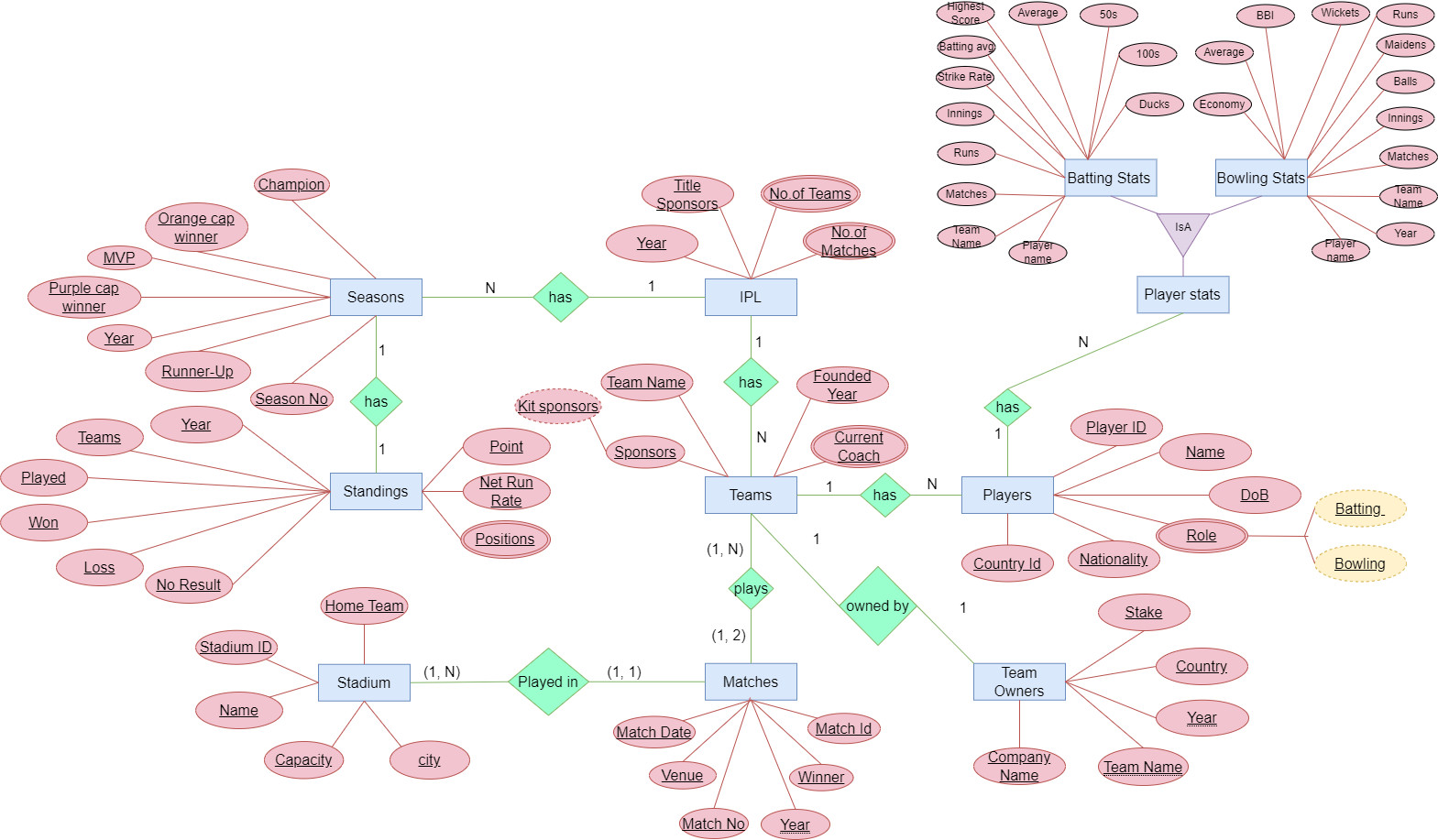
Improvement: our app includes detailed player statistics for both batting and bowling, allowing users to analyze individual player performances across seasons.

**3. User-Friendly Interface:**

Pain Point: Some apps may have cluttered interfaces or lack user-friendly design, making it challenging for users to navigate and find relevant information.

Improvement: Design our app with a clean and intuitive interface, making it easy for users to explore different sections, view stats, and access features without confusion.

# **EER Diagram:**



# **Anomalies:**

Anomalies in the relational model refer to inconsistencies or errors that can arise when working with relational databases, specifically in the context of data insertion, deletion, and modification. There are different types of anomalies that can occur in referencing and referenced relations which can be discussed as:

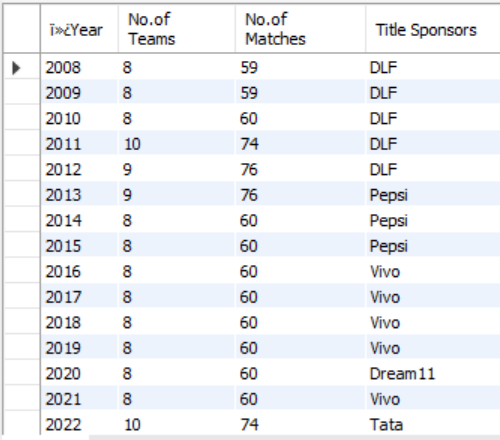
These anomalies can be categorized into three types:

* Insertion Anomalies
* Deletion Anomalies
* Update Anomalies.

**Anomalies which could occur in our project:**

1. **In IPL table:**

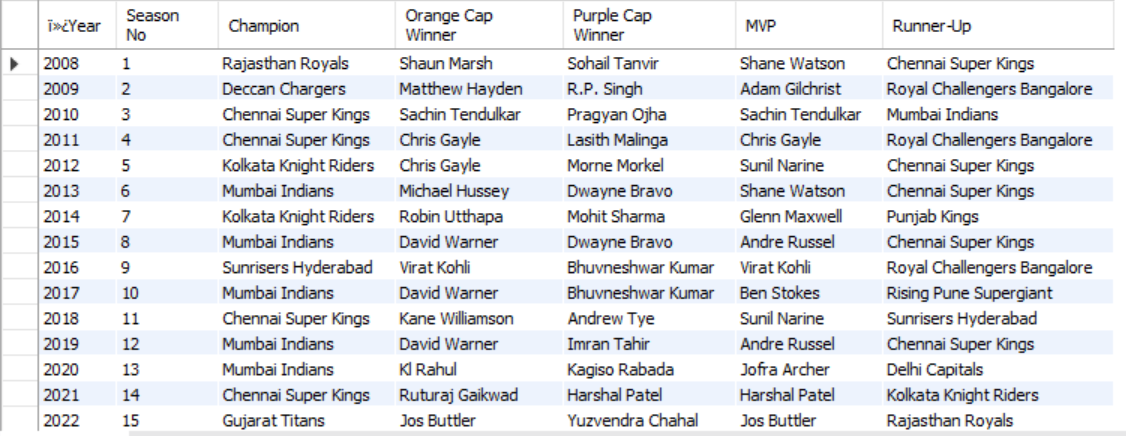
* **Update Anomalies** occurs: If there is a need to update the name of a sponsor (e.g., change "Vivo" to "Vivo India"), it would require updating multiple records, which could lead to inconsistencies if not done carefully.





**2. In seasons table:**

* **Deletion Anomaly occurs**: If a team is removed from the tournament (e.g., due to financial reasons or other factors), deleting the corresponding record would result in the loss of information about players associated with that team, leading to a deletion anomaly.





**3. In Matches Table:**

* **1. Insertion Anomalies** occur: If a new match record is added, but one or more required fields are left empty or NULL, it could lead to an insertion anomaly.

For example, if a match record is inserted without specifying the match date or winner, it would create incomplete data, making it challenging to interpret.

* **Updation Anomalies** occur: Inconsistencies might arise if data is updated in one place and not in all relevant places. For instance:

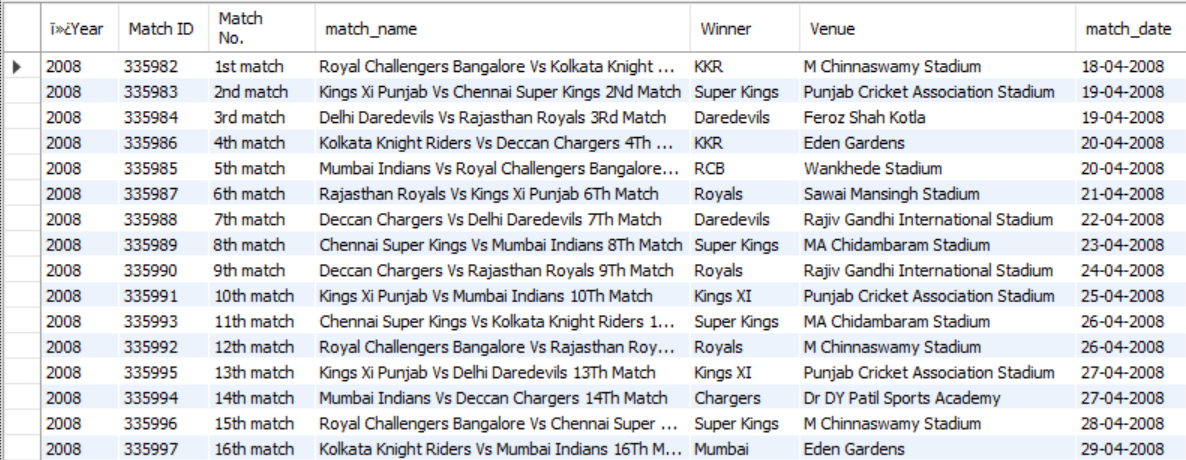
If the venue of a match is updated in one row but not in all rows where the same venue is listed, it could lead to inconsistency.

Updating a team name (e.g., abbreviating "Chennai Super Kings" to "CSK" in some records) inconsistently can cause confusion. Modifying a match date without updating related records can lead to discrepancies.

* **Deletion Anomalies** occur: Removing a match record could inadvertently delete information that might be relevant to other parts of the data. For example:

If a match record is deleted, and it contains the only occurrence of a particular venue or team, it could result in the loss of information about that venue or team.

Deleting a match record without considering its impact on aggregate statistics or historical analysis could lead to data loss.



**4. In Team Owners Table:**

* **Insertion Anomalies** occurs:

In the "Company Name" column: "Red Chillies Entertainment, Mehta Group" and "Manoj Badale, Amisha Hathiramani, Lachlan Murdoch" entries contain multiple entities in a single field, which could lead to insertion anomalies if not properly handled.

* **Update Anomalies** occur:

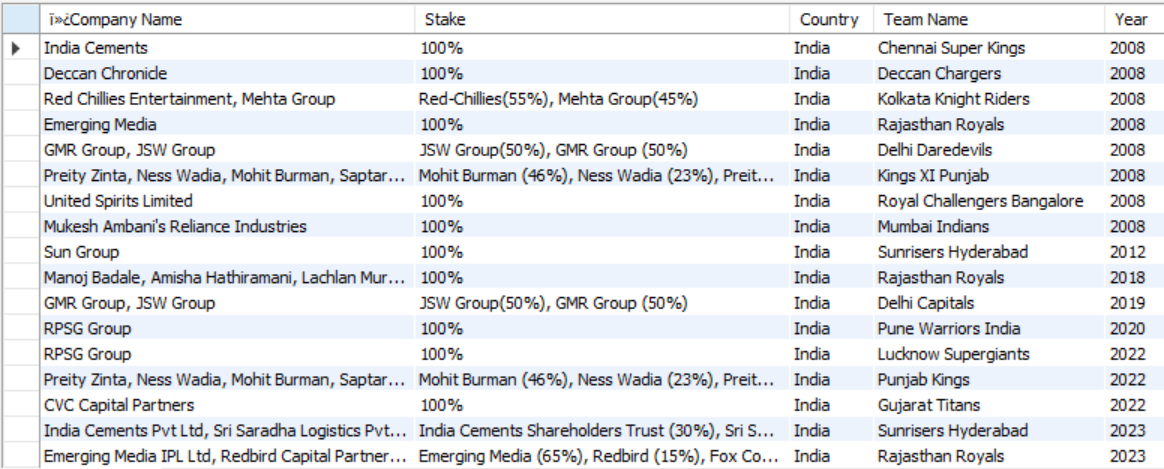
In the "Stake" column: If the stake percentages change over time for a particular company, updating these values could lead to updation anomalies if not updated consistently across all relevant records.

In the "Team Name" column: If a team changes its name (e.g., "Delhi Daredevils" becomes "Delhi Capitals"), updating this information inconsistently could lead to updation anomalies.

* **Deletion Anomalies** occurs:

In the "Company Name" column: If a company exits or is acquired, deleting its record could lead to deletion anomalies if it's the only reference to that company in the dataset.

In the "Team Name" column: If a team is disbanded or rebranded, deleting its record without considering historical data could lead to deletion anomalies.



**5. In Teams Table:**

* **Insertion Anomalies** occur:

In the "Team-Name" column: Entries for "Deccan Chargers," "Rising Pune Supergiant," "Gujarat Lions," "Pune Warriors India," and "Kochi Tuskers Kerala" have missing values (NA) for other attributes. These entries could cause insertion anomalies if additional attributes are added in the future.

* **Updation Anomalies** occur:

In the "Current Coach" column: If the coach of a team changes, updating this information inconsistently could lead to updation anomalies.

In the "Sponsors" and "Kit Sponsors" columns: If a team changes its sponsors or kit sponsors, updating this information inconsistently could lead to updation anomalies.



**6.In stadiums table:**

* **Insertion Anomalies** occur:

In the "Home Team" column: Entries such as "NA" indicate stadiums that don't have a designated home team, potentially leading to insertion anomalies if data dependencies exist that require a home team for each stadium.

* **Updation Anomalies** occur:

In the "Name" column: If a stadium changes its name, updating this information inconsistently across records could lead to updation anomalies.

In the "Home Team" column: If a stadium's home team changes, updating this information inconsistently could lead to updation anomalies.

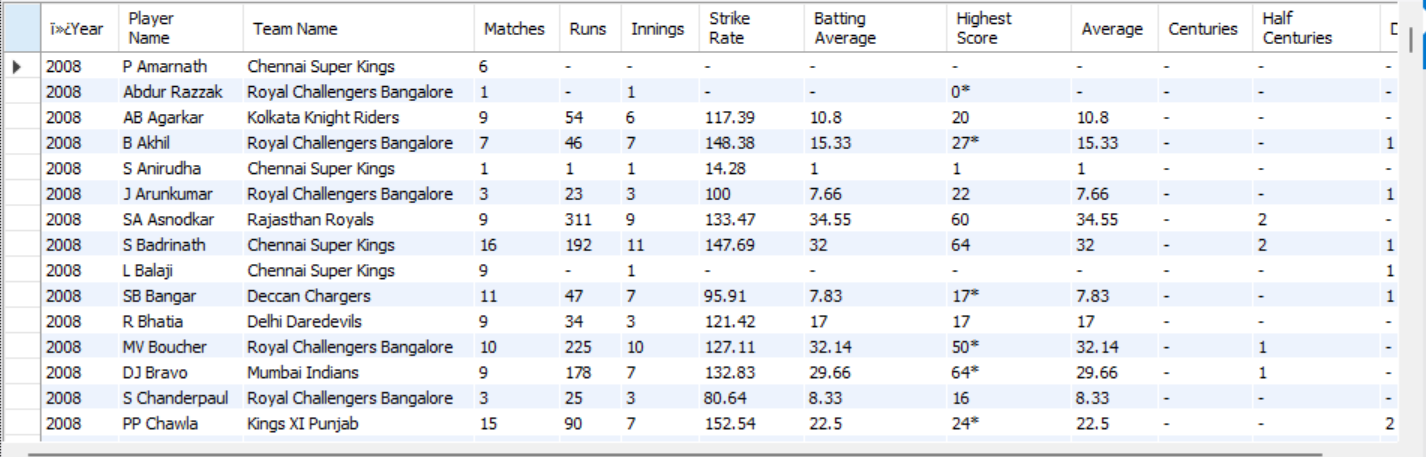


**7. In Player Stats-Batting & Bowling Tables each:**

* **Insertion Anomalies** occur:

An insertion anomaly occurs when certain attributes cannot be inserted into the database without the presence of other attributes. For example:

In 2009, player "Anureet Singh" appears in the Kolkata Knight Riders team but has no data regarding matches, runs, or other statistics. This makes it difficult to insert information about his performance without violating the database's integrity.



**8. In Players Table:**

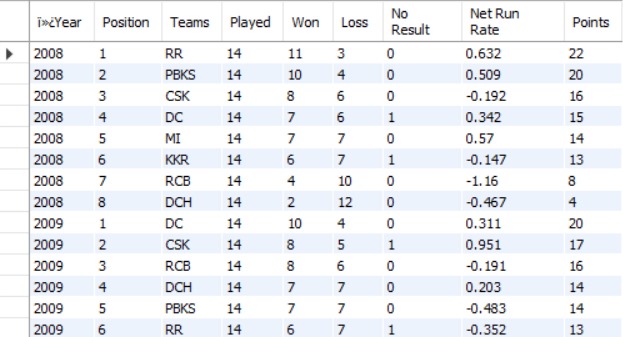
* **Insertion Anomalies** occur:

There are few batters who don’t bowl, so while inserting their bowling style there is an anomaly. Similarly, there are few bowlers who have not batted, so while inserting their batting style there is an anomaly.



**9. In Standings Table:**

No anomalies were found in the Standings Table.



# **Functional Dependencies**

**1. Team-Owners Table:**

Company Name, Team Name → Stake, Country, Year

Year, Stake → Company Name, Country, Team Name

**2. Players Stats-Batting Table:**

Year, Player Name → Team Name, Matches, Runs, Innings, Strike Rate, Batting Average, Highest Score, Average, Centuries, Half Centuries, Ducks

Team Name, Player Name, Matches → Runs, Innings, Strike Rate, Batting Average, Highest Score, Average, Centuries, Half Centuries, Ducks

**3. Players Stats-Bowling Table:**

Year, Player Name → Team Name, Mat, Inns, Balls, Mdns, Runs, Wkts, BBI, Ave, Econ, SR, 5, 10, Ct, St

Team Name, Player Name, Mat → Inns, Balls, Mdns, Runs, Wkts, BBI, Ave, Econ, SR, 5, 10, Ct, St

**4. Players Table:**

Player ID → Name, dob, Bowling, Batting, Nationality, country\_id

Nationality → country\_id

**5. IPL Table:**

Year → No.of Teams, No.of Matches, Title Sponsors

No.of Teams → No.of Matches

**6. Teams Table:**

Team Name → Founded, Current Coach, Sponsors, Kit Sponsors

Founded → Team Name, Current Coach, Sponsors, Kit Sponsors

**7. Standings Table:**

Year, Position → Teams, Played, Won, Loss, No Result, Net Run Rate, Points

Position, Teams → Played, Won, Loss, No Result, Net Run Rate, Points

**8. Matches Table:**

Year, Match ID → Match No., match\_name, Winner, Venue, match\_date

Match No., match\_date → Year, Match ID, match\_name, Winner, Venue

**9. Seasons Table:**

Year, Season No → Champion, Orange Cap Winner, Purple Cap Winner, MVP, Runner-Up

Champion → Year, Season No, Orange Cap Winner, Purple Cap Winner, MVP, Runner-Up

**10. Stadium Table:**

Stadium ID → Name, Home Team, Capacity, City

Home Team → Stadium ID, Name, Capacity, City

# **Normalization:**

Normalization is the process of minimizing redundancy from a relation or set of relations. Redundancy in relation may cause insertion, deletion, and update anomalies. So, it helps to minimize redundancies in relations. Normal forms are used to eliminate or reduce redundancy in database tables.

Important Points Regarding Normal Forms in DBMS

**First Normal Form (1NF):** This is the most basic level of normalization. In 1NF, each table cell should contain only a single value, and each column should have a unique name. The first normal form helps to eliminate duplicate data and simplify queries.

**Second Normal Form (2NF):** 2NF eliminates redundant data by requiring that each non-key attribute be dependent on the primary key. This means each column should be directly related to the primary key, not to other columns.

**Third Normal Form (3NF):** 3NF builds on 2NF by requiring that all non-key attributes are independent of each other. This means each column should be directly related to the primary key, not to any other columns in the same table.

**Boyce-Codd Normal Form (BCNF):** BCNF is a stricter form of 3NF that ensures that each determinant in a table is a candidate key. In other words, BCNF ensures that each non-key attribute is dependent only on the candidate key.

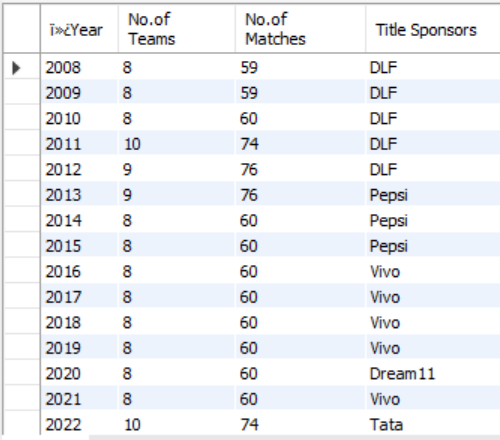
**Fourth Normal Form (4NF):** 4NF is a further refinement of BCNF that ensures that a table does not contain any multi-valued dependencies.

**Fifth Normal Form (5NF):** 5NF is the highest level of normalization and involves decomposing a table into smaller tables to remove data redundancy and improve data integrity.

Normal forms help to reduce data redundancy, increase data consistency, and improve database performance. However, higher levels of normalization can lead to more complex database designs and queries. It is important to strike a balance between normalization and practicality when designing a database.

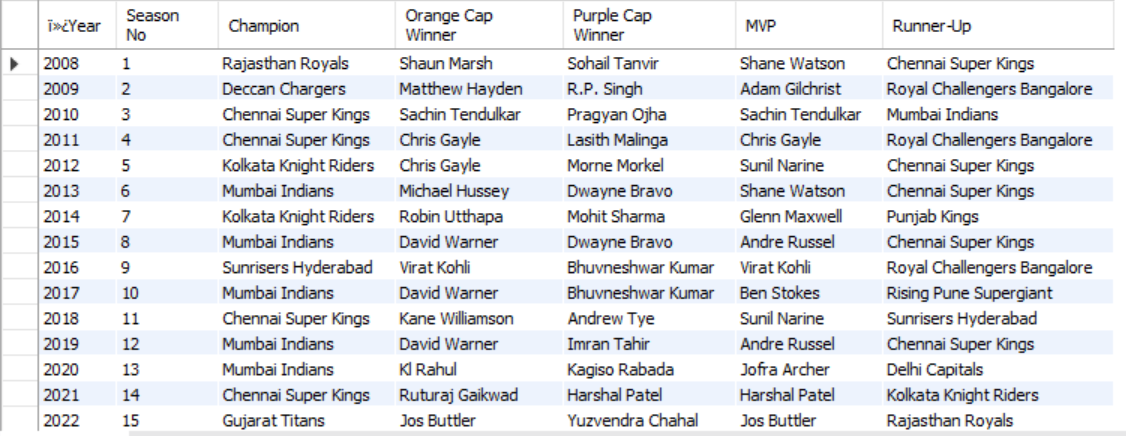
Following are the Examples of Normal forms in our datasets:

**1. For IPL table:**



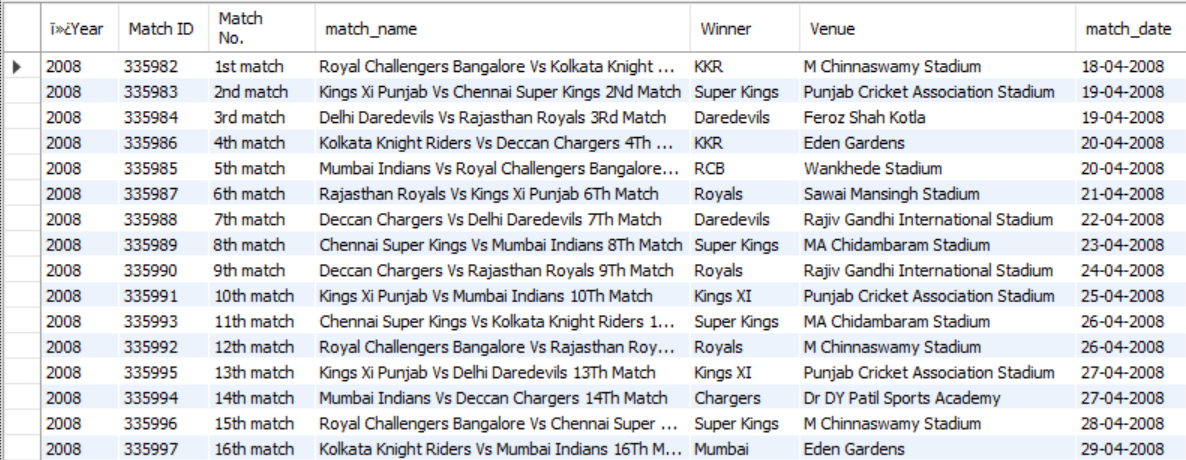
* **First Normal Form (1NF)**: The table is in 1NF because it contains only atomic values, and each column has a single value.
* **Second Normal Form (2NF)**: The table is in 2NF because it's already in 1NF, and there are no partial dependencies. All non-prime attributes depend on the entire primary key (IPL-ID).
* **Third Normal Form (3NF)**: The table is in 3NF because it's in 2NF, and there are no transitive dependencies. Each non-prime attribute depends only on the primary key, and there are no dependencies between non-prime attributes.

**2. For Seasons Table:**



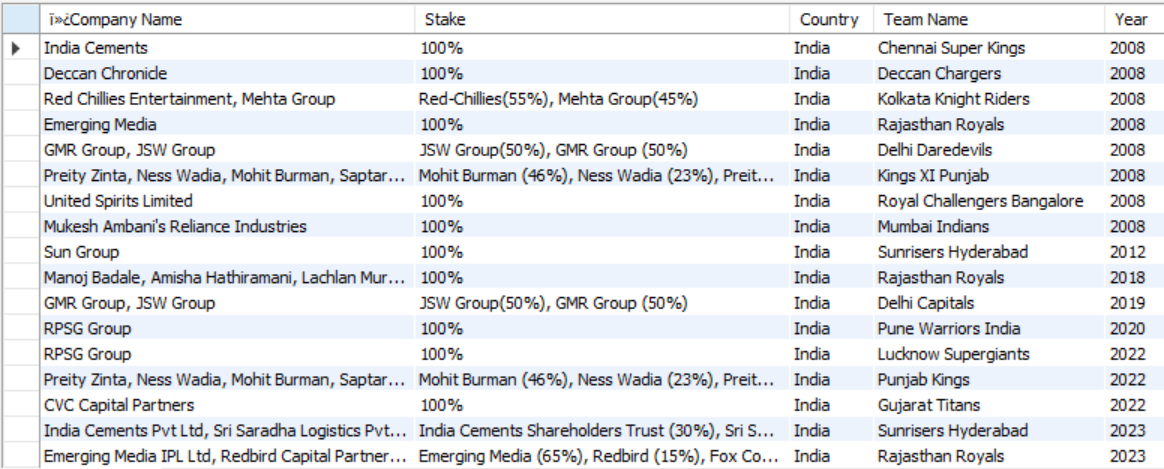
* **First Normal Form (1NF)**: The table appears to be in 1NF as it consists of a two-dimensional table with atomic values in each cell, and there are no repeating groups.
* **Second Normal Form (2NF)**: To be in 2NF, the table must first satisfy 1NF and then ensure that all non-prime attributes are fully functionally dependent on the primary key. In this case, it seems that each column (Year, Season No, Champion, Orange Cap Winner, Purple Cap Winner, MVP, Runner-Up) is dependent on the combination of Year and Season No. Therefore, it satisfies 2NF.
* **Third Normal Form (3NF)**: For 3NF, the table must already be in 2NF and then ensure that there are no transitive dependencies. Looking at the data, it appears that there are no transitive dependencies between non-prime attributes. Each attribute is directly dependent on the primary key (Year, Season No). Thus, it satisfies 3NF.

**3. For Matches table:**



* **First Normal Form (1NF)**: The table appears to be in 1NF as it consists of a two-dimensional table with atomic values in each cell, and there are no repeating groups.
* **Second Normal Form (2NF)**: To be in 2NF, the table must first satisfy 1NF and then ensure that all non-prime attributes are fully functionally dependent on the primary key. In this case, it seems that each column (Year, Match ID, Match No., match\_name, Winner, Venue) is dependent on the combination of Year and Match ID. Therefore, it satisfies 2NF.
* **Third Normal Form (3NF)**: For 3NF, the table must already be in 2NF and then ensure that there are no transitive dependencies. Looking at the data, it appears that there are no transitive dependencies between non-prime attributes. Each attribute is directly dependent on the primary key (Year, Match ID). Thus, it satisfies 3NF.

**4. For Team Owners Table:**



* **First Normal Form (1NF)**: The table appears to be in 1NF as it consists of a two-dimensional table with atomic values in each cell, and there are no repeating groups.
* **Second Normal Form (2NF)**: To be in 2NF, the table must first satisfy 1NF and then ensure that all non-prime attributes are fully functionally dependent on the primary key. In this case, it seems that each column (Company Name, Stake, Country, Team Name, Year) is dependent on the combination of Team Name and Year. However, Stake seems to have multiple values, implying partial dependency. For example, in the case of "Red Chillies Entertainment, Mehta Group", the stake is divided into two distinct parts, indicating a partial dependency on the primary key. Therefore, it does not fully satisfy 2NF.
* **Third Normal Form (3NF)**: For 3NF, the table must already be in 2NF and then ensure that there are no transitive dependencies. Looking at the data, it appears that there are no transitive dependencies between non-prime attributes. Each attribute is directly dependent on the primary key (Team Name, Year). Thus, it satisfies 3NF.

**5. For Teams Table:**



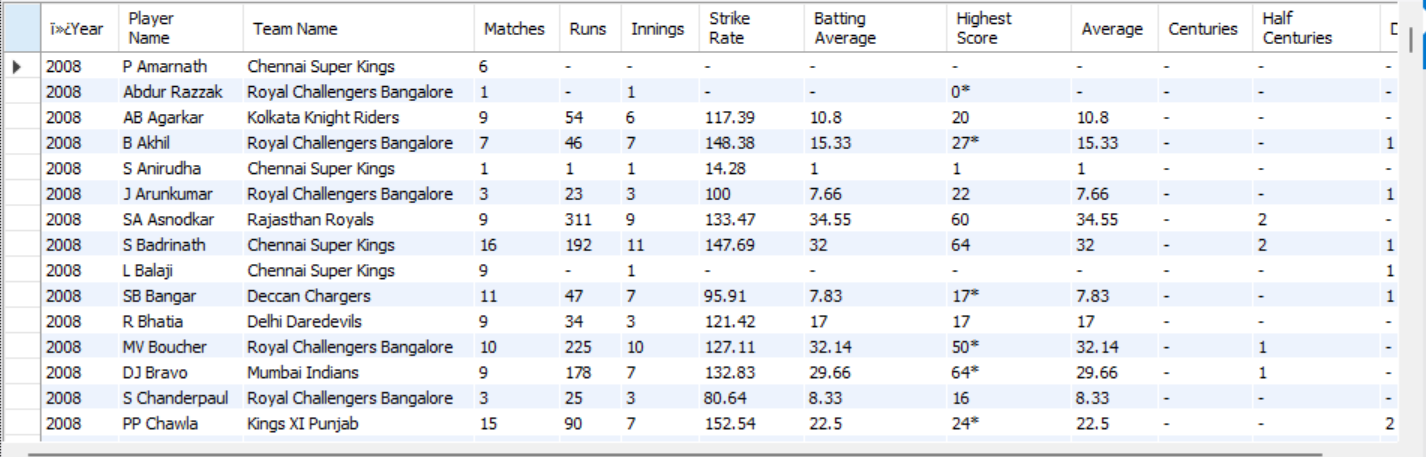
* **First Normal Form (1NF)**: The table seems to be in 1NF because it represents a two-dimensional table with atomic values in each cell, and there are no repeating groups.
* **Second Normal Form (2NF)**: To meet 2NF requirements, the table must satisfy 1NF and ensure that all attributes are fully functionally dependent on the primary key. In this table, "Team-Name" seems to be the primary key. All other attributes (Founded, Current Coach, Sponsors, Kit Sponsors) are dependent on the "Team-Name" attribute. Therefore, it meets the requirements of 2NF.
* **Third Normal Form (3NF)**: For 3NF, the table must already be in 2NF and ensure that there are no transitive dependencies. Looking at the data, there don't seem to be any transitive dependencies. Each non-prime attribute depends directly on the primary key (Team-Name). Therefore, it satisfies the requirements of 3NF.

**6. For Stadium Table:**



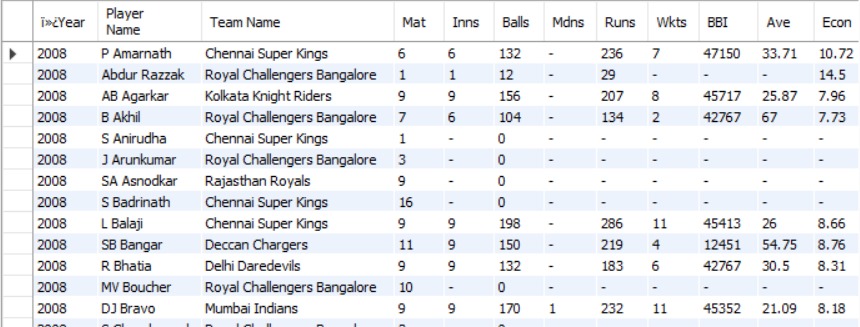
1. **First Normal Form (1NF)**: The table appears to be in 1NF as it represents tabular data with atomic values in each cell, and there are no repeating groups.
2. **Second Normal Form (2NF)**: To meet 2NF requirements, the table must first satisfy 1NF and then ensure that all attributes are fully functionally dependent on the primary key. In this table, "Stadium ID" seems to be the primary key. The attributes "Name", "Home Team", "Capacity", and "City" all depend directly on the primary key. Therefore, it meets the requirements of 2NF.
3. **Third Normal Form (3NF)**: For 3NF, the table must already be in 2NF and then ensure that there are no transitive dependencies. Looking at the data, there don't seem to be any transitive dependencies. Each non-prime attribute depends directly on the primary key (Stadium ID). Therefore, it satisfies the requirements of 3NF.

**7. For Player Stats-Batting Table:**



* **First Normal Form (1NF)**: The table seems to be in 1NF as it represents tabular data with atomic values in each cell. There are no repeating groups, and each attribute contains only single values.
* **Second Normal Form (2NF)**: To satisfy 2NF, the table must first fulfill the requirements of 1NF and ensure that all attributes are fully functionally dependent on the primary key. In this table, "Year" and "Player Name" together can act as a composite primary key. However, some attributes like "Matches," "Runs," "Innings," "Strike Rate," "Batting Average," "Highest Score," "Average," "Centuries," "Half Centuries," and "Ducks" depend solely on the combination of "Year" and "Player Name." Therefore, it meets the requirements of 2NF.
* **Third Normal Form (3NF)**: For 3NF, the table must already be in 2NF and ensure that there are no transitive dependencies. Looking at the data, there don't seem to be any transitive dependencies. Each non-prime attribute depends directly on the composite primary key ("Year" and "Player Name"). Therefore, it satisfies the requirements of 3NF.

**8. For Player Stats-Bowling Table:**



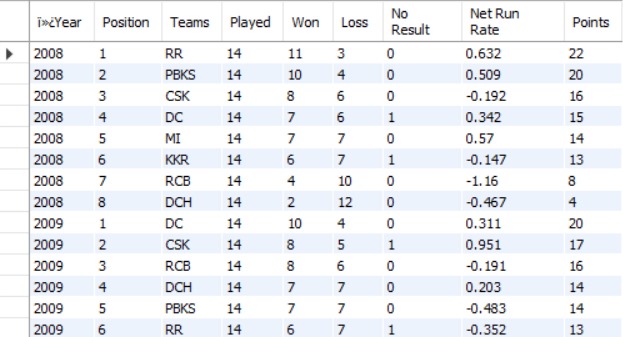
* **First Normal Form (1NF)**: The table appears to be in 1NF as it represents tabular data with atomic values in each cell. There are no repeating groups, and each attribute contains only single values.
* **Second Normal Form (2NF)**: To satisfy 2NF, the table must first fulfill the requirements of 1NF and ensure that all attributes are fully functionally dependent on the primary key. In this table, "Year," "Player Name," and "Team Name" together can act as a composite primary key. However, some attributes like "Mat," "Inns," "Balls," "Mdns," "Runs," "Wkts," "BBI," "Ave," "Econ," "SR," "5," "10," "Ct," and "St" depend solely on the combination of "Year," "Player Name," and "Team Name." Therefore, it meets the requirements of 2NF.
* **Third Normal Form (3NF)**: For 3NF, the table must already be in 2NF and ensure that there are no transitive dependencies. Looking at the data, there don't seem to be any transitive dependencies. Each non-prime attribute depends directly on the composite primary key ("Year," "Player Name," and "Team Name"). Therefore, it satisfies the requirements of 3NF.

**9. For Players Table:**



* **First Normal Form (1NF)**: The table appears to be in 1NF as it represents tabular data with atomic values in each cell. Each attribute contains only single values, and there are no repeating groups.
* **Second Normal Form (2NF)**: To satisfy 2NF, the table must first fulfill the requirements of 1NF and ensure that all attributes are fully functionally dependent on the primary key. In this table, "Player ID" can serve as a primary key, and "Name" can be considered a candidate key. All other attributes like "dob," "Bowling," "Batting," "Nationality," and "country\_id" seem to be fully functionally dependent on the primary key. Therefore, it meets the requirements of 2NF.
* **Third Normal Form (3NF)**: For 3NF, the table must already be in 2NF and ensure that there are no transitive dependencies. Looking at the data, there don't seem to be any transitive dependencies. Each non-prime attribute depends directly on the primary key ("Player ID"). Therefore, it satisfies the requirements of 3NF.

10. **For Standings Table:**

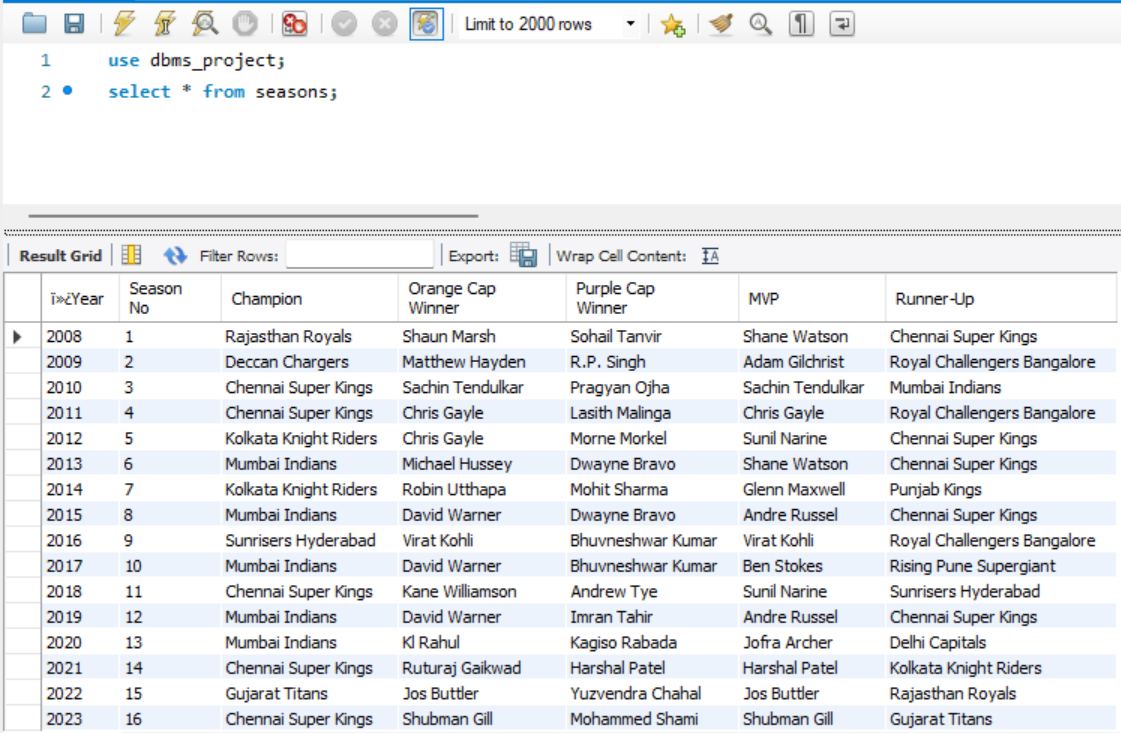


* **First Normal Form (1NF)**: The table appears to be in 1NF as it represents tabular data with atomic values in each cell. Each attribute contains only single values, and there are no repeating groups.
* **Second Normal Form (2NF)**: For a table to be in 2NF, it must first satisfy 1NF and have all non-prime attributes fully functionally dependent on the primary key. In this table, the primary key seems to be a composite key consisting of "Year" and "Position." All other attributes (Teams, Played, Won, Loss, No Result, Net Run Rate, Points) appear to be functionally dependent on this composite key. Therefore, the table satisfies 2NF.
* **Third Normal Form (3NF)**: To achieve 3NF, a table must first be in 2NF and ensure that there are no transitive dependencies. Looking at the provided data, there don't seem to be any transitive dependencies. Each non-prime attribute depends directly on the composite primary key (Year, Position). Thus, the table satisfies 3NF.

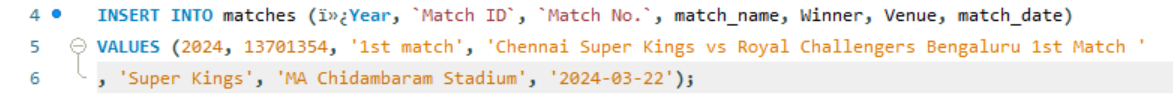
# **Query Implementation:**

# **Basic Queries:**

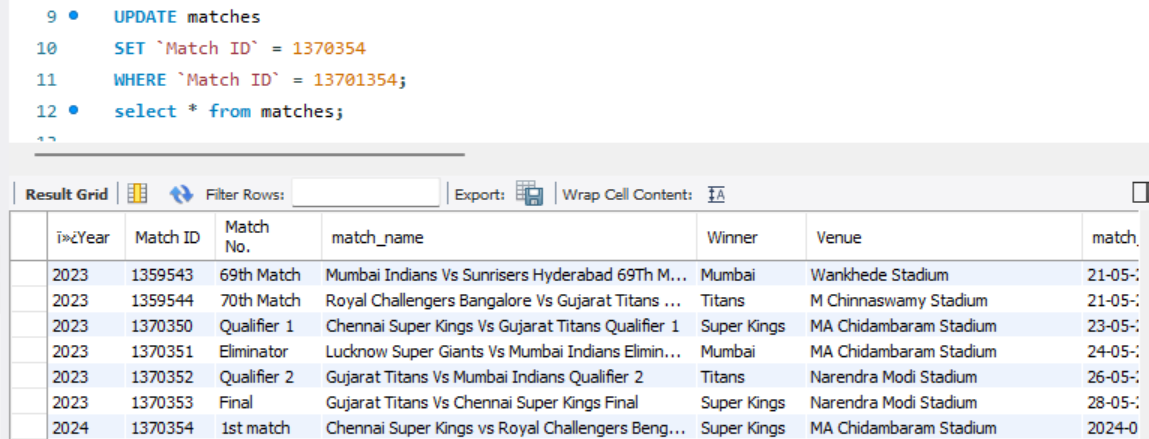
**1. Select Query:**



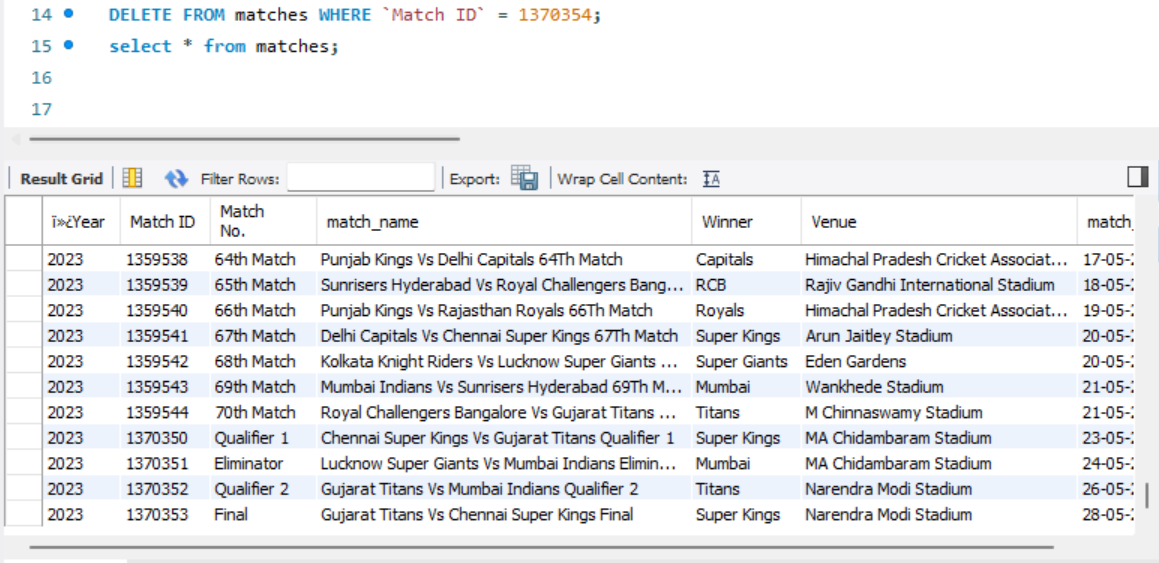
**2. Insert Query:**



**3. Update Query:**

**4. Delete Query:**





# **Joins:**

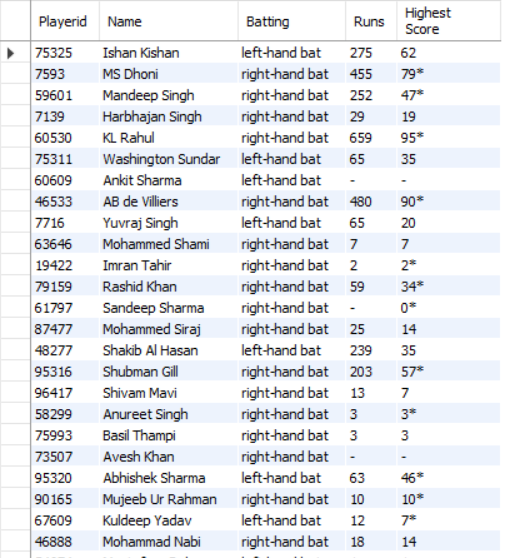
**1. Displaying the performances of all the players in the year 2018:**

select P.Playerid,p.Name,p.Batting,Ba.Runs as 'Runs',Ba.`Highest Score` as 'Highest Score'

from players p

join battingstats Ba on Ba.`Player Name`=P.name

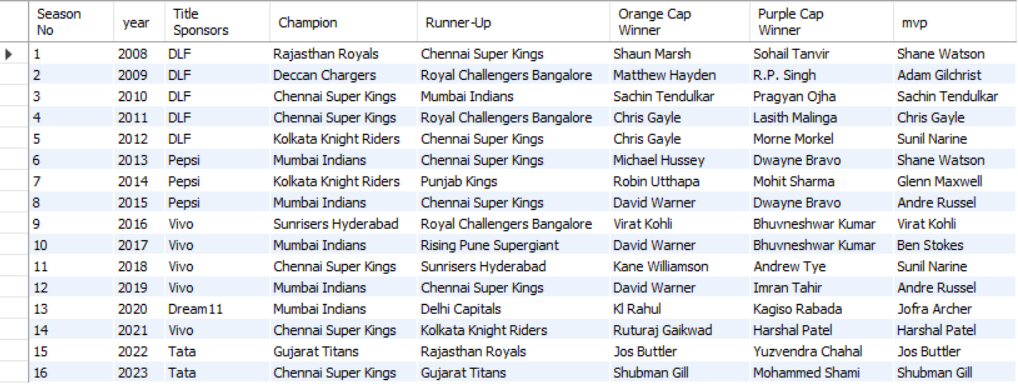
where ba.Year=2018;



**2. Showing the champions, runners up, MVP, Orange and purple cap winners for every season:**

Select s.`Season No`, i.year,i.`Title Sponsors`, S.Champion, S.`Runner-Up`,S.`Orange n S.Year=i.year; Cap Winner`,S.`Purple Cap Winner`,S.mvp from ipl i join Seasons S o

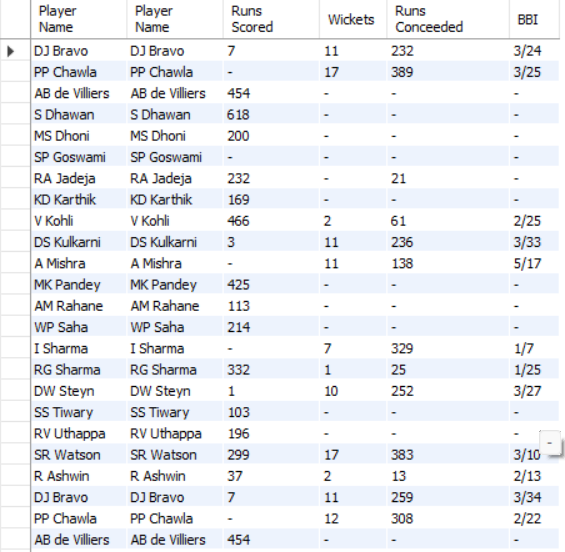
Output:



**3. Displaying the allrounder stats of year 2020:**

Select ba.`Player Name`,Ba.`Player Name`,Ba.Runs as 'Runs Scored',Bo.wkts as 'Wickets',Bo.runs as 'Runs Conceeded',Bo.BBI from battingstats ba right join bowlingstats bo on bo.`Player Name`=Ba.`Player Name` where Ba.year=2020;

Output:



**4. The highest score and best figures of bowlers who have at least taken a wicket in the year 2023:**

SELECT

bo.`Player Name`,

Bo.BBI as 'Best Figures',

ba.`Highest Score`

FROM

bowlingstats bo

right JOIN

battingstats ba ON ba.`Player Name` = bo.`Player Name`

WHERE

bo.year = 2023 and bo.BBI!='-';

Output:



**5. Batting stats of the purple cap winner of 2019:**

SELECT

s.`Season No`,

s.Year,

s.`Purple Cap Winner`,

bs.`Team Name`,

bs.Matches,

bs.Runs,

bs.Innings,

bs.`Strike Rate`,

bs.`Batting Average`,

bs.`Highest Score`,

bs.Average,

bs.Centuries,

bs.`Half Centuries`,

bs.Ducks

FROM

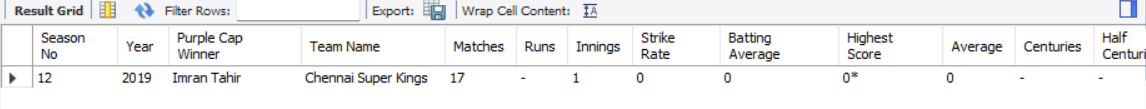
seasons s

left JOIN

battingstats bs ON bs.`Player Name` = s.`Purple Cap Winner` AND bs.Year = s.Year

where bs.year = 2019;

Output:



# Views:

**1. A view to display all the Chennai Super Kings Bowlers:**

create view SuperKingsBowlers AS

Select `Player Name`,year,`Team Name`,Inns,WKTS

from bowlingstats

where `Team Name`='Chennai Super Kings';

select \* from SuperKingsBowlers;

Output:



**2. View to display the centuries scored by the DeccanChargers Batsmen across all seasons:**

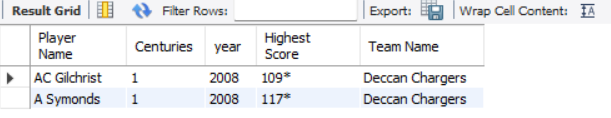
create view DeccanChargersCenturies as

select `Player Name`,Centuries,year,`Highest Score`,`Team Name`

from battingstats where `Team Name` = 'Deccan Chargers' and Centuries>=1;

select \* from DeccanChargersCenturies;

Output:



**3. View to display all the matches won by RCB in 2012:**

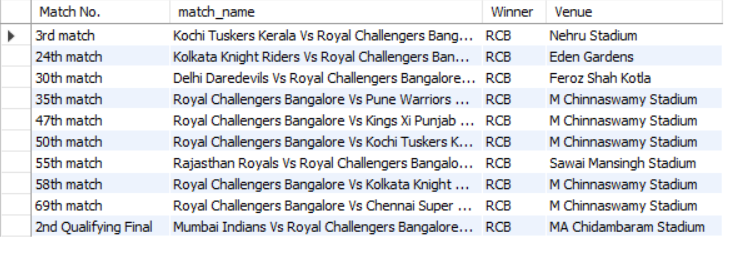
create view RCBWins2012 as

select `Match No.`,match\_name,Winner,Venue

from matches where year=2011 and winner = 'RCB';

select \* from RCBWins2012;

Output:



# **Function:**

**1. Function to find which team has finished at top position in Standings table:**

DELIMITER //

CREATE FUNCTION TopTeamMostTimes() RETURNS VARCHAR(255)

DETERMINISTIC

BEGIN

DECLARE top\_team VARCHAR(255);

SELECT Teams INTO top\_team

FROM (

SELECT Teams, COUNT(\*) AS num\_times\_top

FROM standings

WHERE Position = 1

GROUP BY Teams

ORDER BY num\_times\_top DESC

LIMIT 1

) AS top\_team\_count;

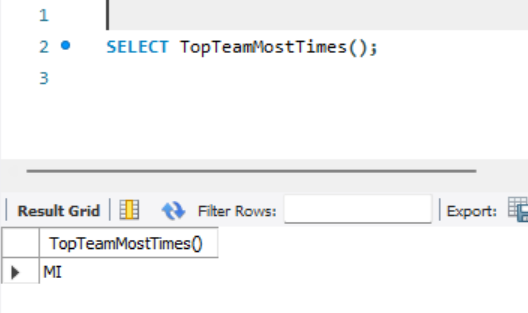
RETURN top\_team;

END//

DELIMITER ;

SELECT TopTeamMostTimes();

Output:



**2. Function to get most successful team in ipl:**

DELIMITER //

CREATE FUNCTION MostSuccessfulTeam()

RETURNS VARCHAR(100)

DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE champion\_team VARCHAR(100);

-- Retrieve the team that became the champion most times

SELECT Champion INTO champion\_team

FROM (

SELECT Champion, COUNT(\*) AS num\_championships

FROM seasons

GROUP BY Champion

ORDER BY num\_championships DESC

LIMIT 1

) AS most\_successful\_team;

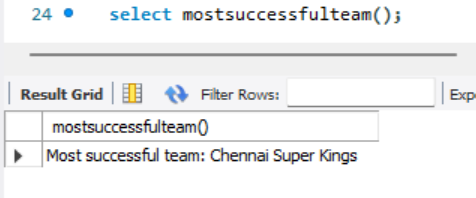
RETURN CONCAT('Most successful team: ', champion\_team);

END//

DELIMITER ;

select mostsuccessfulteam();

Output:



**3. Function to get teams which has been runner-up most time:**

DELIMITER //

DROP FUNCTION IF EXISTS MostRunnerUpTeam//

CREATE FUNCTION MostRunnerUpTeam() RETURNS VARCHAR(255)

DETERMINISTIC

BEGIN

DECLARE runner\_up\_team VARCHAR(255);

SELECT `runner-up` INTO runner\_up\_team

FROM (

SELECT `runner-up`, COUNT(\*) AS num\_times\_runner\_up

FROM seasons

GROUP BY `runner-up`

ORDER BY num\_times\_runner\_up DESC

LIMIT 1

) AS runner\_up\_count;

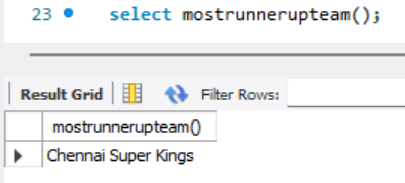
RETURN runner\_up\_team;

END//

DELIMITER ;

select mostrunnerupteam();

Output:



# **Stored Procedures**

**1. Stored Procedure to see which team has won most matches across different seasons of the IPL:**

DELIMITER $$

CREATE PROCEDURE MostWinsTeam()

BEGIN

SELECT Winner, COUNT(\*) AS Wins

FROM matches

GROUP BY Winner

ORDER BY Wins DESC

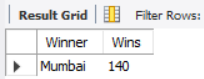
LIMIT 1;

END$$

DELIMITER ;

CALL MostWinsTeam();

Output:



**2. Stored Procedure to see the player with most centuries across all seasons:**

DELIMITER $$

CREATE PROCEDURE MostCenturies()

BEGIN

SELECT `Player Name`, SUM(Centuries) AS TotalCenturies

FROM `player stats-batting`

GROUP BY `Player Name`

ORDER BY TotalCenturies DESC

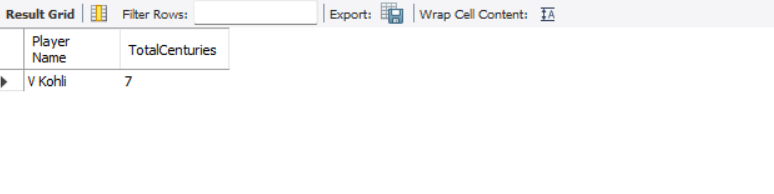
LIMIT 1;

END$$

DELIMITER ;

call MostCenturies();

Output:



**3. Stored Procedure to see the player with most half centuries across all seasons:**

DELIMITER $$

CREATE PROCEDURE MostHalfCenturies()

BEGIN

SELECT `Player Name`, SUM(`Half Centuries`) AS TotalHalfCenturies

FROM `player stats-batting`

GROUP BY `Player Name`

ORDER BY TotalHalfCenturies DESC

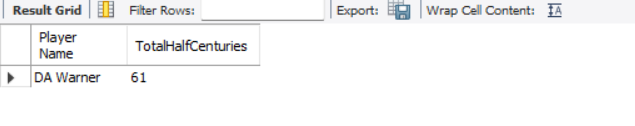
LIMIT 1;

END$$

DELIMITER ;

call MostHalfCenturies();

Output:



**4. Stored Procedures for finding the top 5 players with most 5 wicket hauls:**

DELIMITER $$

CREATE PROCEDURE MostFiveWicketHauls()

BEGIN

SELECT `Player Name`, SUM(`5`) AS FiveWicketHauls

FROM `player stats-bowling`

GROUP BY `Player Name`

ORDER BY FiveWicketHauls DESC

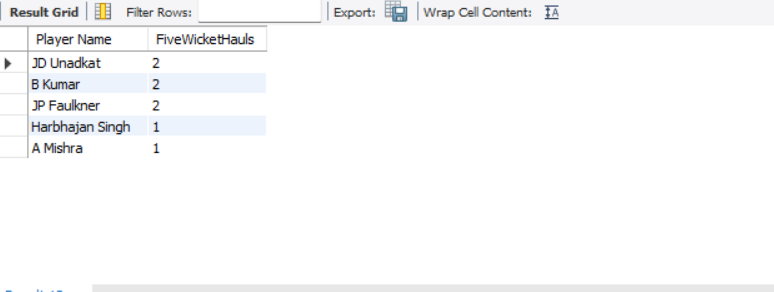
LIMIT 5;

END$$

DELIMITER

call MostFiveWicketHauls();

Output:



**5. Stored procedure for finding the top 5 players with most maidens:**

DELIMITER $$

CREATE PROCEDURE MostMaidens()

BEGIN

SELECT `Player Name`, SUM(Mdns) AS Maidens

FROM `player stats-bowling`

GROUP BY `Player Name`

ORDER BY Maidens DESC

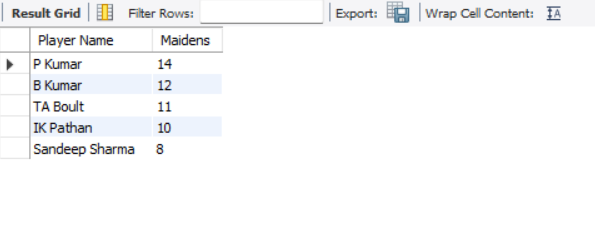
LIMIT 5;

END$$

DELIMITER

call MostMaidens();

Output:



# **Triggers**

**1. MostRuns Trigger:**

This trigger displays the most runs scored by a batter after each insert operation is executed on the `player stats-batting` table. The result is then inserted into another table for logging purposes since a trigger cannot return a set result directly.

Code:

DELIMITER $$

CREATE TRIGGER DisplayMostRuns

AFTER INSERT ON `player stats-batting`

FOR EACH ROW

BEGIN

DECLARE most\_runs\_player VARCHAR(255);

DECLARE most\_runs INT;

SELECT `Player Name`, SUM(Runs) INTO most\_runs\_player, most\_runs

FROM `player stats-batting`

GROUP BY `Player Name`

ORDER BY most\_runs DESC

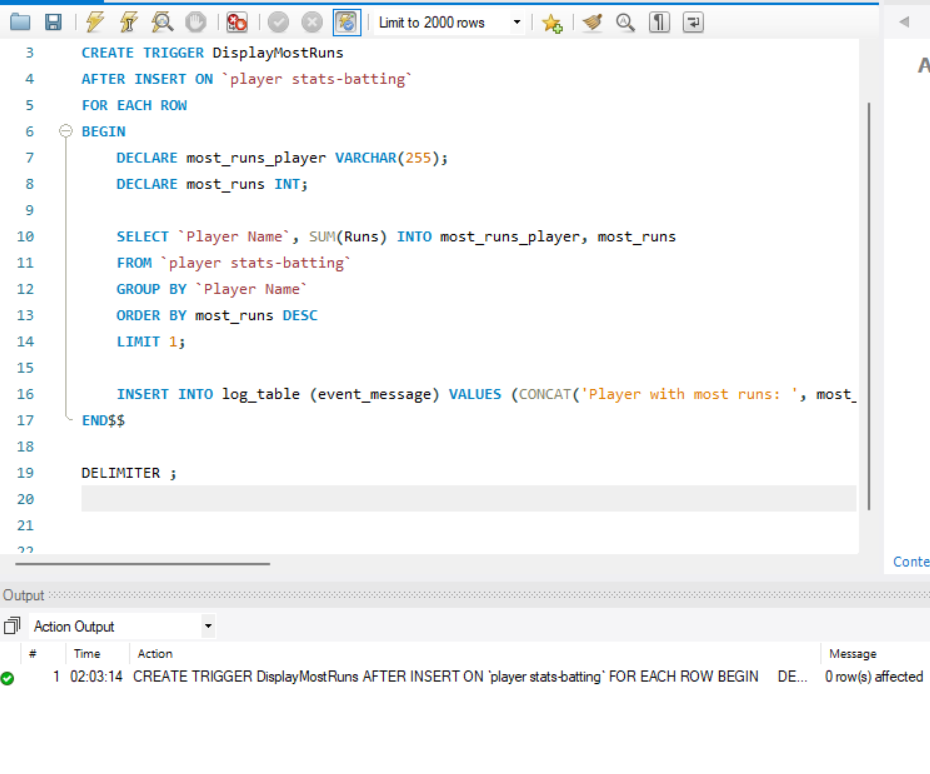
LIMIT 1;

INSERT INTO log\_table (event\_message) VALUES (CONCAT('Player with most runs: ', most\_runs\_player, ', Runs: ', most\_runs));

END$$

DELIMITER ;

OUTPUT:



2. **Trigger to display latest champion:**

This trigger is used to display the latest champion once an insert operation is performed on the table Seasons. The trigger inserts the latest champion into the LatestChampion table after each insert operation on the Seasons table. If a record for the latest champion already exists in the LatestChampion table, the trigger updates the existing record with the new champion.

Code:

DELIMITER $$

CREATE TRIGGER TriggerDisplayLatestChampion

AFTER INSERT ON Seasons

FOR EACH ROW

BEGIN

DECLARE latest\_champion VARCHAR(255);

SELECT Champion INTO latest\_champion

FROM Seasons

ORDER BY Year DESC

LIMIT 1;

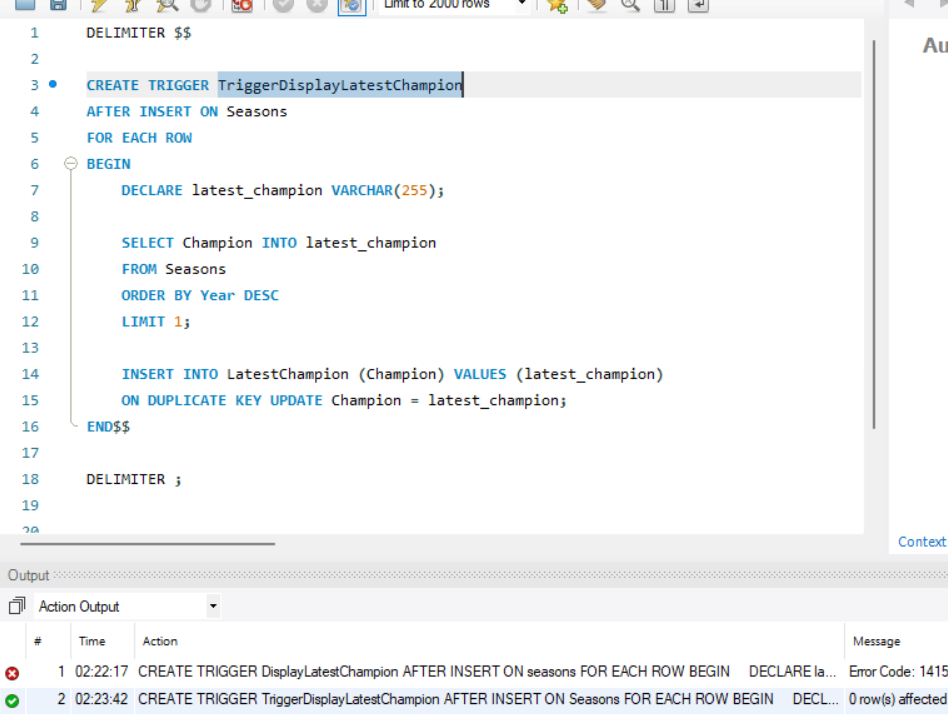
INSERT INTO LatestChampion (Champion) VALUES (latest\_champion)

ON DUPLICATE KEY UPDATE Champion = latest\_champion;

END$$

DELIMITER ;

OUTPUT:



**3. DisplayMostWickets Trigger:**

This trigger is used to display the player with most wickets after the insert function. The result is stored in a session variable most\_wickets\_result. After the result is executed, you can fetch the result by calling the session variable:

SELECT @most\_wickets\_result;

Code:

DELIMITER $$

CREATE TRIGGER DisplayMostWickets

AFTER INSERT ON `player stats-bowling`

FOR EACH ROW

BEGIN

DECLARE most\_wickets\_player VARCHAR(255);

SELECT `Player Name` INTO most\_wickets\_player

FROM `player stats-bowling`

ORDER BY Wkts DESC

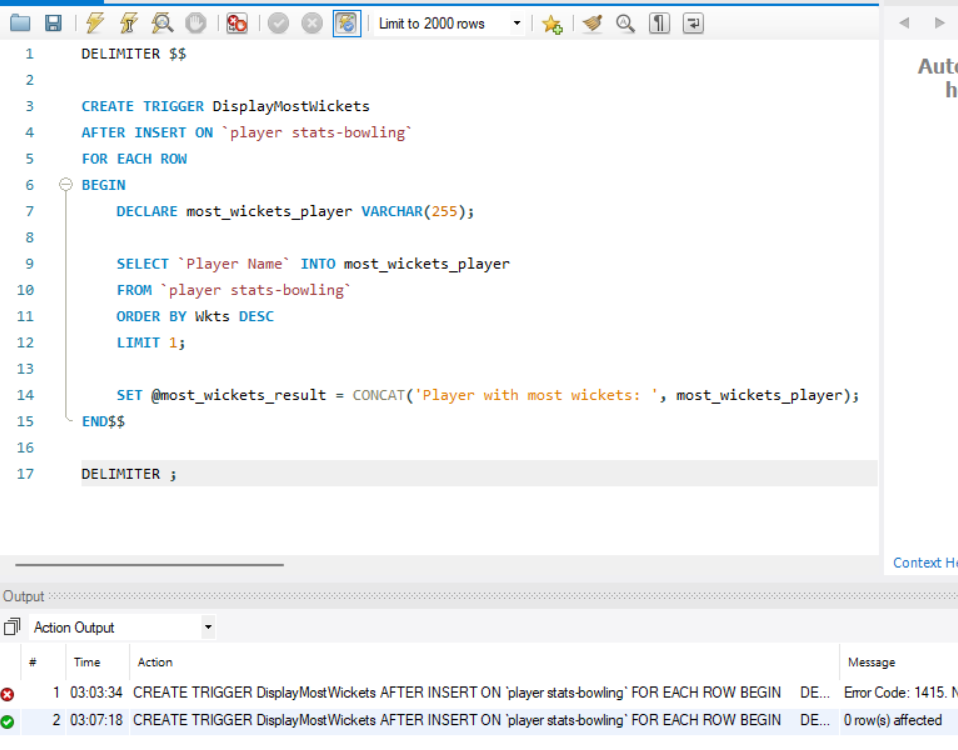
LIMIT 1;

SET @most\_wickets\_result = CONCAT('Player with most wickets: ', most\_wickets\_player);

END$$

DELIMITER ;

OUTPUT:



# **References:**

Espncricinfo: <https://www.espncricinfo.com/>

# **Conclusion:**

The "IPL Stats Management" project has been a comprehensive endeavor aimed at organizing and managing the vast array of data associated with the Indian Premier League (IPL). Through the integration of a relational database management system (DBMS) and Java programming, we have successfully designed and implemented a robust platform capable of storing, retrieving, and analyzing detailed statistics from all IPL seasons to date.

Throughout the development process, we have addressed various aspects crucial to the efficiency and effectiveness of the system. Anomalies within the data were identified and rectified, ensuring data integrity and reliability. Normalization techniques were applied to optimize the database structure, reducing redundancy and minimizing the risk of data anomalies. Functional dependencies were carefully analyzed and implemented to maintain consistency and accuracy in the stored data.

The implementation of queries, functions, triggers, and stored procedures has enhanced the system's functionality, allowing for seamless data manipulation and retrieval operations. These features enable users to extract meaningful insights from the extensive IPL dataset efficiently.

Furthermore, the Entity-Relationship (EER) diagram provides a visual representation of the database schema, offering a clear understanding of the relationships between different entities and attributes within the system.

In summary, the IPL Stats Management System provides detailed and correct stats of each and every season played till now.