DATABASE SYSTEMS PROJECT

CRICKET MATCH MANAGEMENT SYSTEM

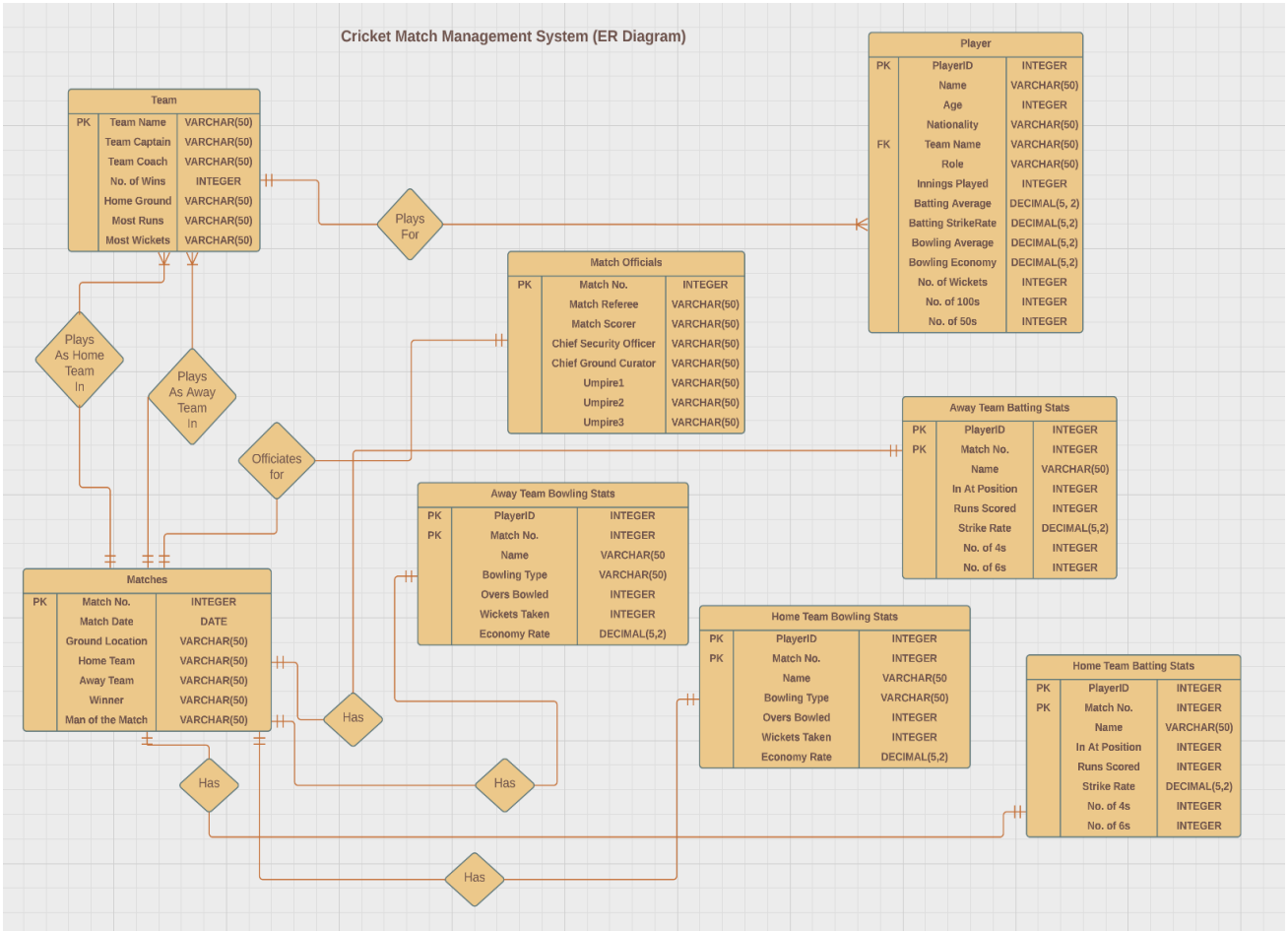
Prepared For:

Database Management Course Ins

1) ***Entity-Relationship Diagram (ER Diagram):***

An Entity-Relationship (ER) diagram is a graphical representation of entities and their relationships to each other, typically used in database design. The diagram consists of entities (objects or concepts), attributes (properties or characteristics of the entities), and relationships (associations between entities).

ER diagrams are used to model the relationships between data in a database system, allowing developers to visualize the structure of the database and ensure that all necessary data is included. They are commonly used in the planning and development stages of a software project, as they provide a clear and concise way of communicating the design of a database to other members of the development team. e In an ER diagram, entities are represented as rectangles, with attributes listed within the rectangle. Relationships are represented as lines connecting the entities, with cardinality (the number of entities involved in the relationship) indicated on the line.



In the above ER Diagram for our project, Cricket Match Management System, we have created the database for a seasonal T20 League, like IPL etc. We have created 8

entities and established relationships between them. The created entities are Match, Team, Player, Match Officials, Home Team Batting Stats, Home Team Bowling Stats, Away Team Batting Stats, and Away Team Bowling Stats.

In the Matches Entity, we have 7 attributes. They are Match Number, Match Date, Ground Location, Home Team Name, Away Team Name, Match Winner, and the Man of the Match. We have considered that each ground location has a unique name. Another assumption that we have taken is that no two matches can occur on the same ground location on the same day. Thus, the set of candidate keys for this entity is *{Match Number, (Match Date, Ground Location)}.* We have made Match Number as the primary key.

In the Team Entity, we have 7 attributes again. They are Team Name, Team Captain, Team Coach, No. of Wins, Home Ground, Most Runs, and Most Wickets. Since we have assumed that Home Ground name is unique, we can agree that Home Ground is a candidate key. Thus, Home Ground and Team Name attributes are candidate keys here, and we have chosen Team Name as the primary key.

For the Player Entity, we have 12 attributes. They are as follows: PlayerID, Name, Age, Nationality, Team Name, Role, Innings played, Batting Average, Batting Strike Rate, Bowling Average, Bowling Economy Rate, No. of Wickets, No. of 100s, No. of 50s. Here, we have assigned PlayerID as the primary key. We also have Team Name as a foreign key attribute which links Player entity and Team entity.

Next, we have Match Officials entity that has 8 attributes. They are Match Number, Match Referee, Match Scorer, Chief Security Officer, Chief Ground Curator, Umpire1, Umpire2, Umpire3. Match Number has been assigned as the primary key for this entity. There are three umpires as two will be on the ground and one will be the TV-Umpire.

We now come to the final 4 entities. These are Home Team Batting Stats, Home Team Bowling Stats, Away Team Batting Stats, and Away Team Bowling Stats.
Home Team and Away Team Batting Stats have the same set of 8 attributes. They are Match Number, PlayerID, Name, In At Position, Runs Scored, Strike Rate, No. of 4s, No. of 6s. For both these entities, we have made the set of Match Number + PlayerID as the primary key.

Similarly, Home Team and Away Team Bowling Stats have the same set of 7 attributes. They are Match Number, PlayerID, Name, Bowling Type, Overs Bowled, No. of Wickets Taken, and Economy Rate. Again, for both these entities, we have made the set of Match Number + PlayerID as the primary key.

Then we talk about Relationships between Entities. There are mainly three types of relationships: One to One relationship, One to Many Relationships, and Many to Many Relationships.

In our ER Diagram, there is a one-to-one relationship between the Match entity and the Match Officials entity, as well as between the Match entity and all the Team Stats entities (such as Home Team Batting Stats). Additionally, we have One-to-Many Relationships between Match and Team entities, and between Team and Player entities. This means that a match can have more than one team playing it, but a team can only play in one match at a time. Similarly, a team can have many players, but a player can belong to only one team at a time.

We have named the relationship between Team and Player as 'Plays For'. Similarly, we have named the relationship between Match and Team as 'Plays In'. For the relationship between Match and Match Officials, we have named it 'Officiates For'. For all the relationships between Match and the team stats (such as Home Team Batting Stats), we have named them as 'has' relationships.

2) ***Conversion of ER Diagram to Relational Schema:***

A relational schema is a way of representing the structure of a relational database using a set of relations or tables, which are made up of rows and columns. Each row in a table represents a single instance or record, and each column represents a specific attribute or field of that record.

The relational schema is used to define the organization and structure of data in a database, and it forms the basis for the actual creation of the database. It includes information about the tables, their attributes or fields, and the relationships between them.

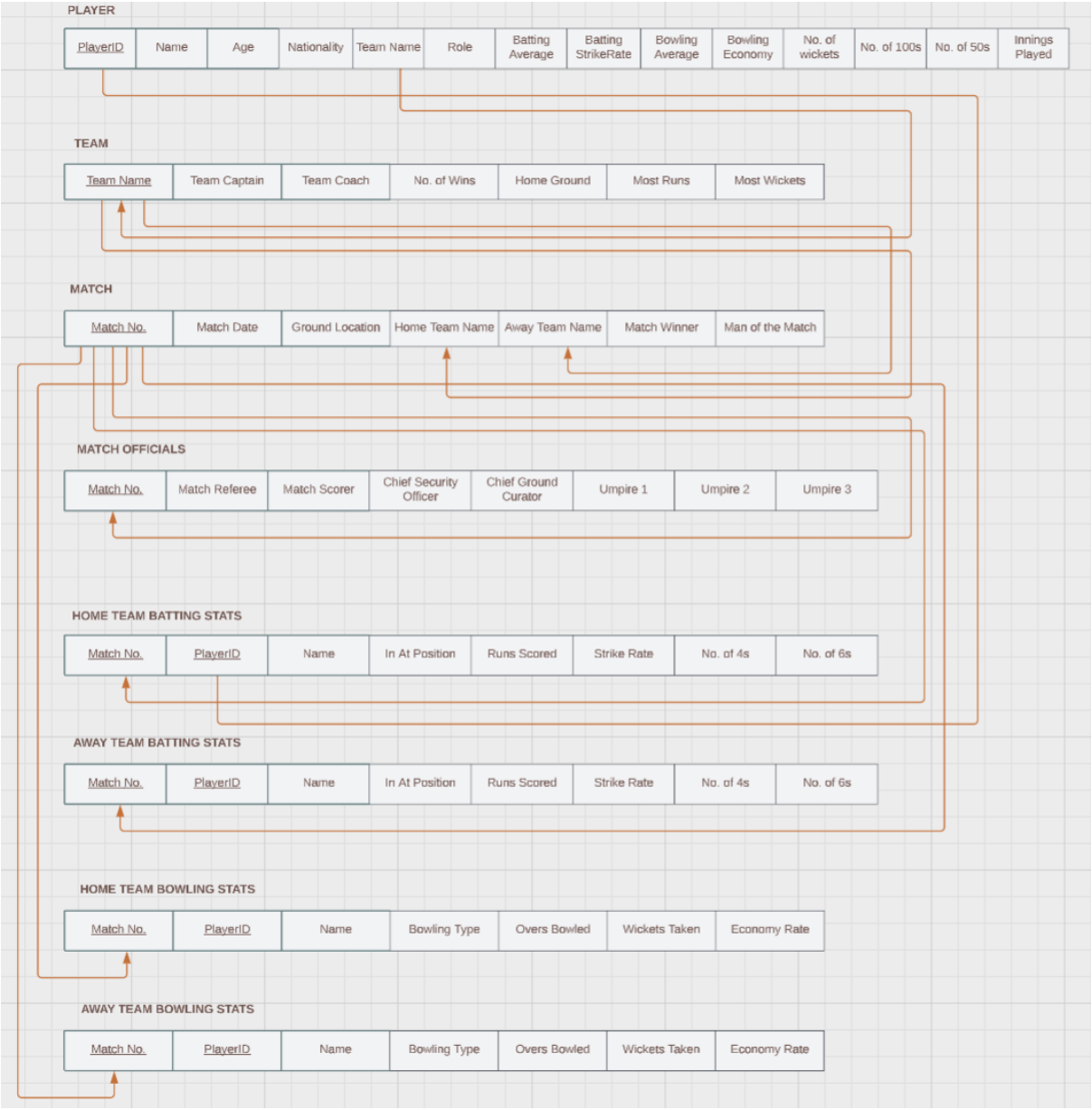
In a relational schema, each table is defined by its name, and a list of its attributes, along with the data types and any constraints that apply to each attribute. Relationships between tables are defined using foreign keys, which link records in one table to records in another.

The relational schema is an essential part of any database design project, as it provides a blueprint for the database structure that developers can follow when building the actual database. It also helps to ensure that the data is organized in a consistent and logical manner, making it easier to access and maintain over time.

In our project, we have a few foreign keys which are used to link the tables in relational schema. For example, in Player entity, we have Team Name as the foreign key, and this can be used to link the player entity with the Team entity. Similarly, we link the match entity with the team stats entites using the Match Number attribute.

Since there is no relation between Match and Player entities in the original ER diagram, we do not make a connection between the Match and Player table here in the relational model.

Shown below is our attempt to convert the ER diagram mentioned above into a relational model.



3) ***Normalization of the Tables to 3NF Form:***

Normalization is a technique used in database design to eliminate data redundancy and dependency. It involves breaking down a table into smaller tables and defining relationships between them to reduce data duplication and ensure data consistency.

The process of normalization involves organizing the data in a database into tables that conform to a set of normal forms. There are several normal forms, ranging from the first normal form (1NF) to the fifth normal form (5NF), each with its own set of rules and requirements. But we generally check only up to the third normal form (3NF).

Functional dependency is a relationship between two attributes in a relational database table. It describes how the value of one attribute is determined by the value of another attribute.

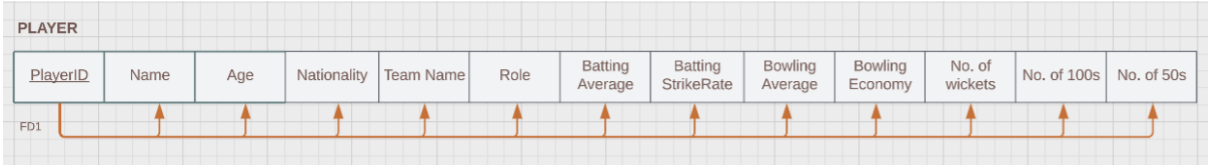
Formally, if A and B are attributes in a table, then we say that B is functionally dependent on A if and only if each value of A is associated with exactly one value of B. In other words, if we know the value of A for a particular record, then we can determine the value of B for that record.

The first normal form (1NF) requires that each table has a primary key and that all columns in the table contain atomic values (values that cannot be further subdivided). The second normal form (2NF) requires that all non-key attributes are functionally dependent on the entire primary key. The third normal form (3NF) requires that all non-key attributes are not functionally dependent on other non-key attributes.

The process of normalization helps to eliminate data redundancy and inconsistencies, which can lead to data inconsistencies and anomalies. By breaking down tables into smaller tables and defining relationships between them, we can ensure that each piece of data is stored only once and that any changes made to the data are consistent across all tables.

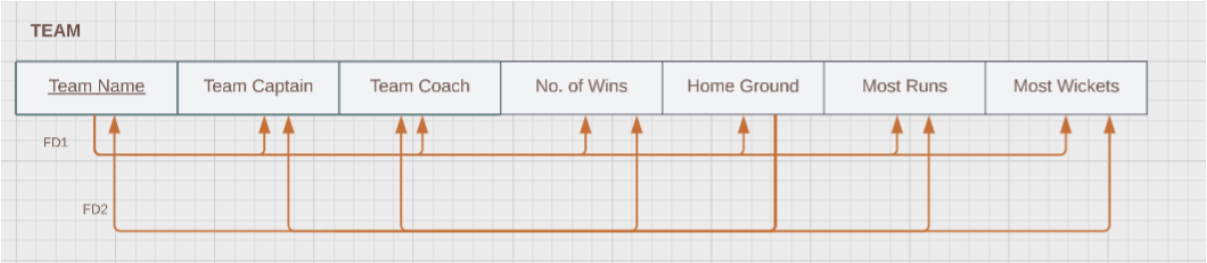
a) Listing all the functional dependencies:

For Player Table:



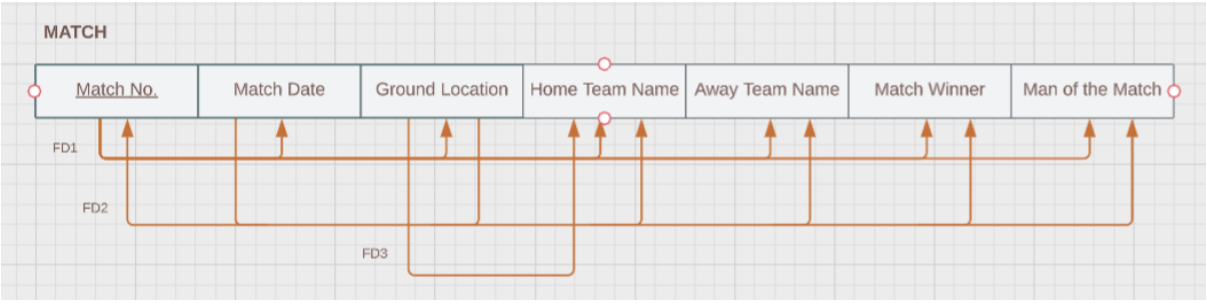
There is only one functional dependency in this table as all the other attributes of this table can be determined by the primary key, which is PlayerID. This is like a default Functional Dependency. No other attributes are functionally dependent here.

For Team Table:



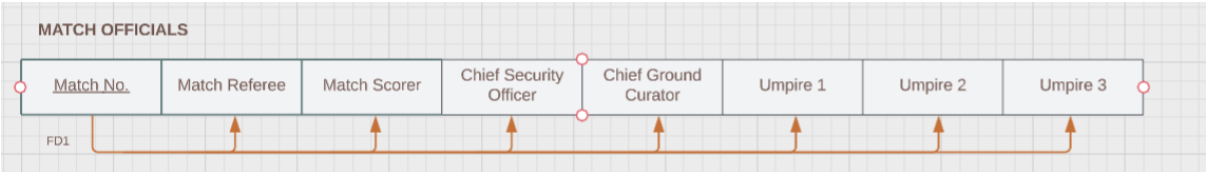
There are two Functional Dependencies present, namely FD1 and FD2. The first functional dependency is the default one in which all the remaining attributes are dependent on the Team Name primary key attribute. The second dependency is between all the other attributes and the Home Ground attribute. Here, we have assumed Home Ground to be unique. Hence, this becomes a candidate key for us, and FD2 functional dependency exists.

For Match Table:



There are three functional dependencies in this *Matches* table, FD1, FD2, and FD3. The first functional dependency is due to the primary key which is the default one. Next, we have the functional dependency where all the other attributes are dependent on the set of Match Date and Ground Location attributes, which is Candidate Key. Once you know the ground location and match date, you can determine the match number, home and away teams, etc. The third and final functional dependency is between the ground location and the home team name. Once you know the ground location, you can determine the home team assigned to that ground.

For Match Officials:



Here, no other set of functional dependencies is possible except the default one. Hence, FD1 is the only functional dependency for the match officials' table.

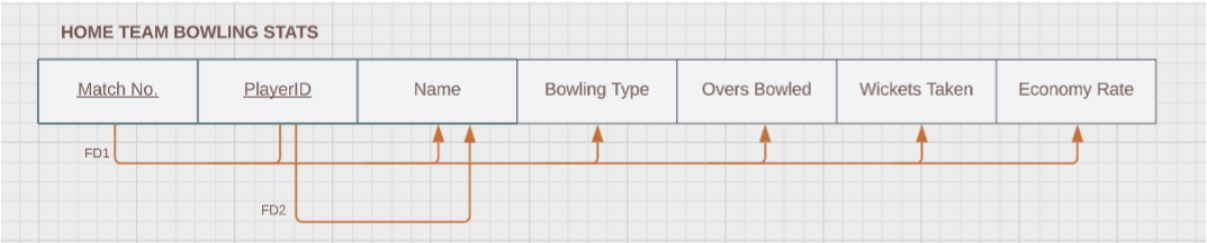
For Home Team Batting Stats Table:

In this table, along with the default primary key functional dependency, which is the set of Match Number and PlayerID in this case, we also have a functional dependency between PlayerID and Name. This is because once you know the PlayerID of a player, you can determine the name of that player. Hence, the name is dependent on the ID. So, a total of 2 functional dependencies are present in this table.

For Away Team Batting Stats Table:

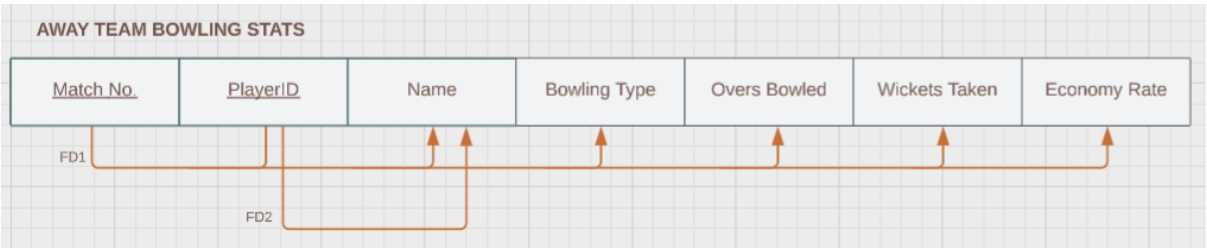
Like the previous table, we have two functional dependencies. The first one is the default primary key functional dependency, and the second one is between PlayerID and Name.

For Home Team Bowling Stats Table:



Here as well, we have 2 functional dependencies. The primary key set is again the match number plus PlayerID. The default functional dependency is the one where all the other attributes are dependent on the primary key. The next one is the dependency between PlayerID and Name. As mentioned above, once you have the PlayerID, it is easy to determine the name of that player and hence the dependency.

For Away Team Bowling Stats:



Like the previous table, this one also has two functional dependencies. Along with the default one, there exists another functional dependency between PlayerID and the name of the player.

b) Conversion to 1NF Form:

For all the above tables, there is no attribute that is multi-valued in nature. This means that all the above tables contain attributes that are atomic (indivisible further). So, we can say from the above arguments that all the tables in this relational schema are in 1NF form by default.

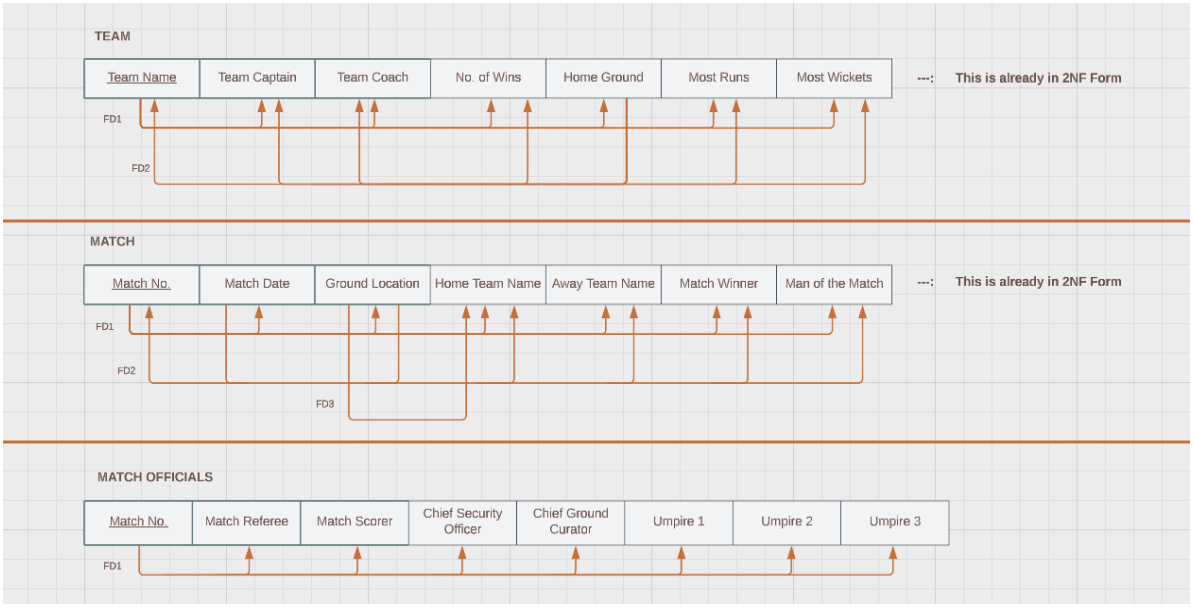
c) Conversion to 2NF Form:

For the Player Table:

Since the primary key is a single attribute and not a set of attributes, we can say that there are no non-key attributes that are dependent *partially* on the primary key. Hence, this table is in 2NF form as well.

For the Team, Matches, and Match Officials Tables:

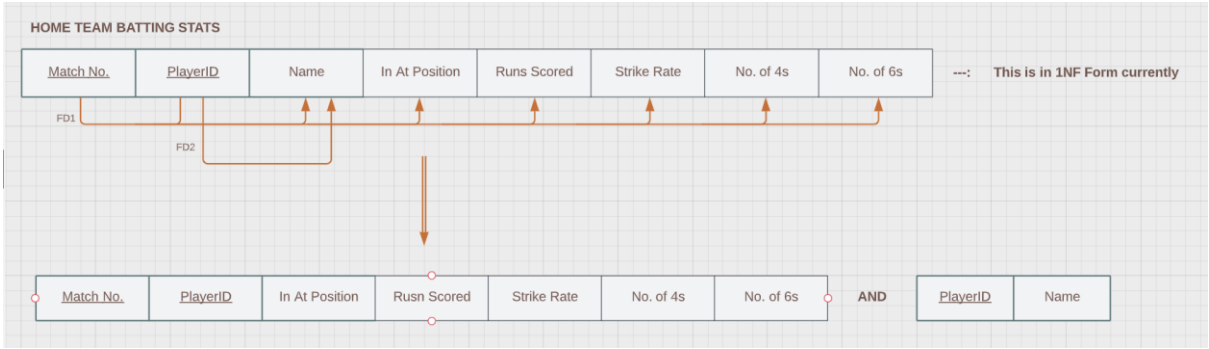
Following the above discussion, we can again see that the primary key in all these tables is a single attribute and not a set of attributes. We can see that there are no non-key attributes that are dependent *partially* on the primary key. Hence, these tables are in 2NF form as well.



For the Home Team Batting Stats Table:

Here, we see that the primary key is a set of two attributes, namely Match Number and PlayerID. We also see that there is a functional dependency in this table that involves only one attribute of the primary key, hence making the non-key attribute partially dependent on the primary key. This violates the 2NF form rule.

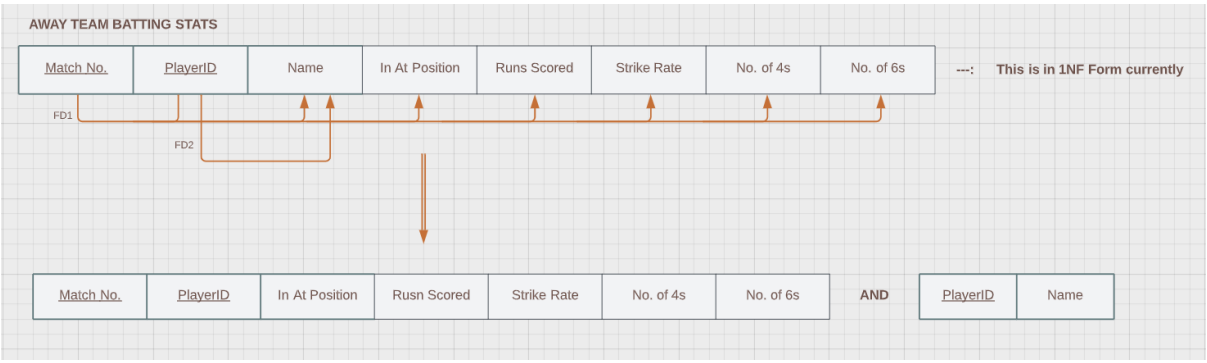
Hence, we need to split up this table into two smaller tables with PlayerID as the common attribute. This will ensure that the new split tables are in 2NF form. Hence, the splitting will happen as follows:



For Away Team Batting Stats Table:

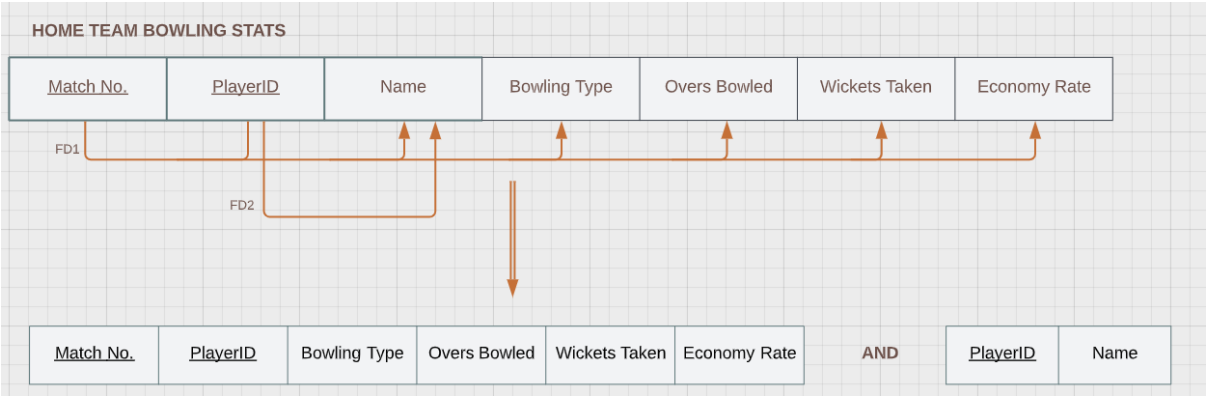
Like the previous case, here as well we have a primary key that is a set of more than 1 attribute. We also have a functional dependency that is partially dependent on the primary key. Hence, to make this into 2NF form, we split the table into 2 smaller tables with PlayerID being the shared attribute.

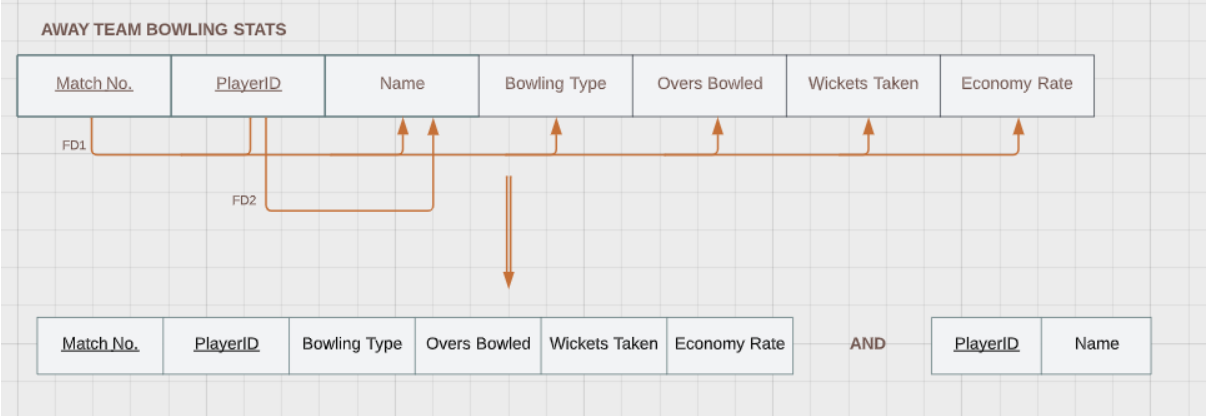
Below shown is the splitting of the table into 2 smaller tables:



For the Home and Away Teams Bowling Stats Table:

Using the above statement, we find that there is a partially dependent functional dependency between PlayerID and Name. So, we split these tables into two smaller tables as follows:

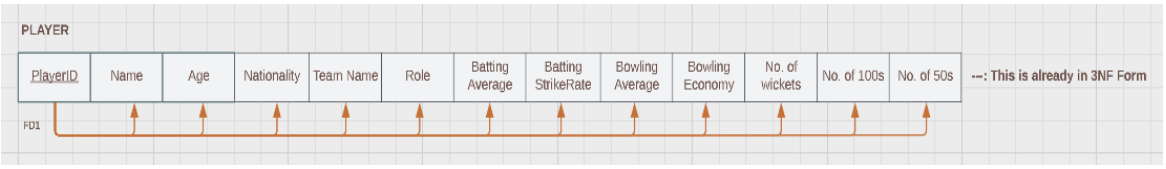




d) ***Conversion to 3NF Form:***

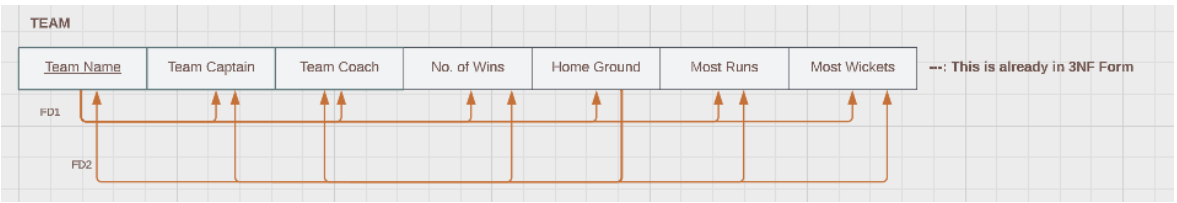
For the Player Table:

For this table, we only have one Functional dependency and that is the default one which includes the primary key. Since there is no transitive functional dependency in this table. Hence, we can conclude that this table is in 3NF form by default.



For the Team Table:

For this table, we have two functional dependencies. To check whether a functional dependency, say A -> B, is in 3NF form or not, we must either ensure that A is a super key or B is a prime attribute. In both cases, we have a candidate key on the left-hand side of the functional dependency. Team Name is a primary key and Home Ground is a candidate key as mentioned earlier in this document. This argument verifies that the *Team* table is in 3NF form by default.



For the Matches Table:

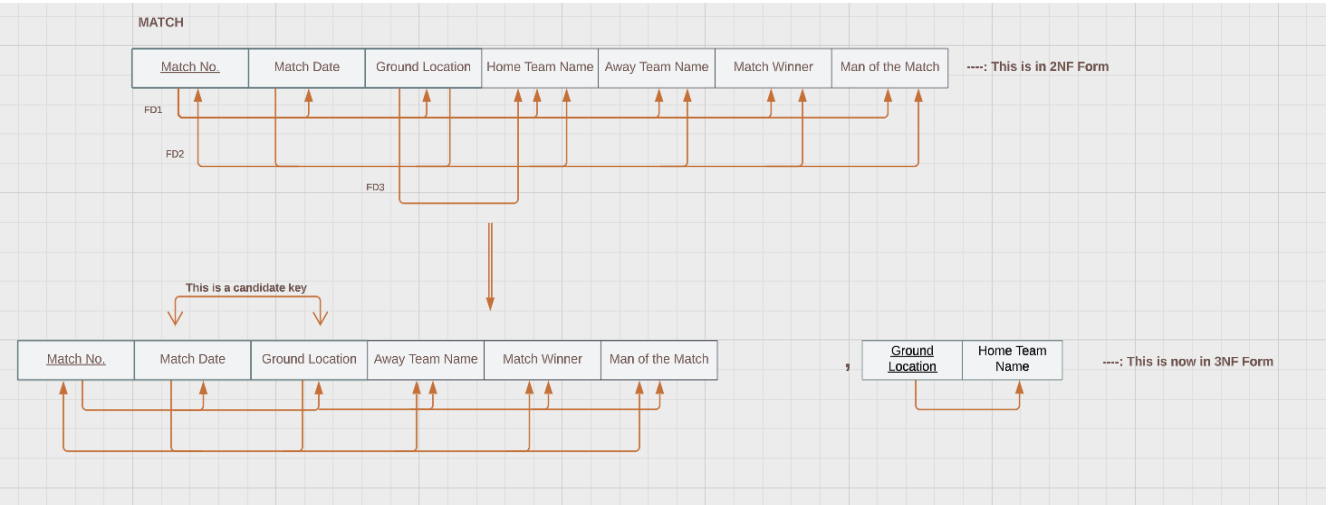
This table is not in 3NF form by default because there exist transitive functional dependencies in this table. The default functional dependency, which includes only the primary key, and the functional dependency which depends on the other candidate key, which is Match Date + Ground Location, do not create a problem. This is because both these functional dependencies have a super key on the left-hand side which is enough to verify that they are in 3NF form.

But the third functional dependency creates a problem for us. We have a functional dependency between Ground Location and Home Team Name. In this, none of them is a super key or a prime attribute. Hence, this functional dependency is not in 3NF Form. Thus, we must resolve this issue to make this table into a 3NF Form.

First, we assign Alphabet codes to these attributes so that it will be easier to represent them. Suppose we assign A to Match Number, B to Match Date, C to Ground Location, D to Home Team Name, E to Away Team Name, F to the Match Winner and G to the Man of the Match. Then we can represent the functional dependencies as A -> BCDEFG, BC ->ADEFG, and C -> D. The last functional dependency creates a problem, so we will split the tables accordingly.

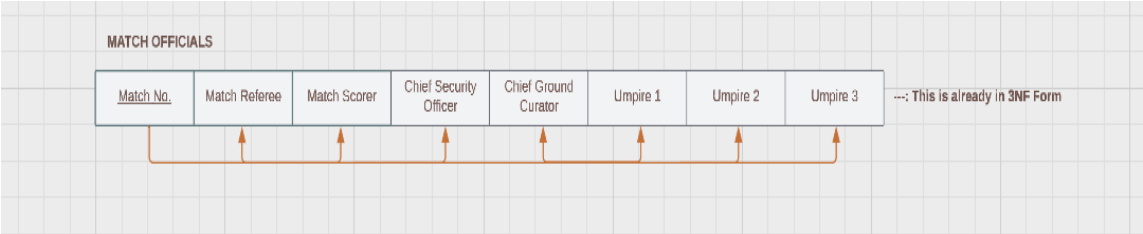
First, we create a table T_1(<u>A</u>BCEFG) where A -> BCEFG and BC ->AEFG functional dependencies are present. In this, A is a primary key, and BC is a candidate key. Next, we create a table T_2(<u>C</u>D) where C -> D functional dependency is present. T_1 table now doesn't create a problem as there are no transitive dependencies in that table. Similarly, T_2 table doesn't create a problem too, since we have made C to be the primary key in this table and C -> D now does not create any issue.

We have split the original *Matches* table into two tables and eliminated the transitive functional dependency problem. Thus, since we have successfully eliminated the transitive dependencies from this table, we can say that we have successfully converted the Match Table into 3NF form. The final tables are T_1, T_2.



For the Match Officials Table:

There exists only one functional dependency in this table, which is the default dependency including the primary key. There is no chance of a transitive dependency to occur. Thus, we can conclude from the above argument that the table is in 3NF form by default.

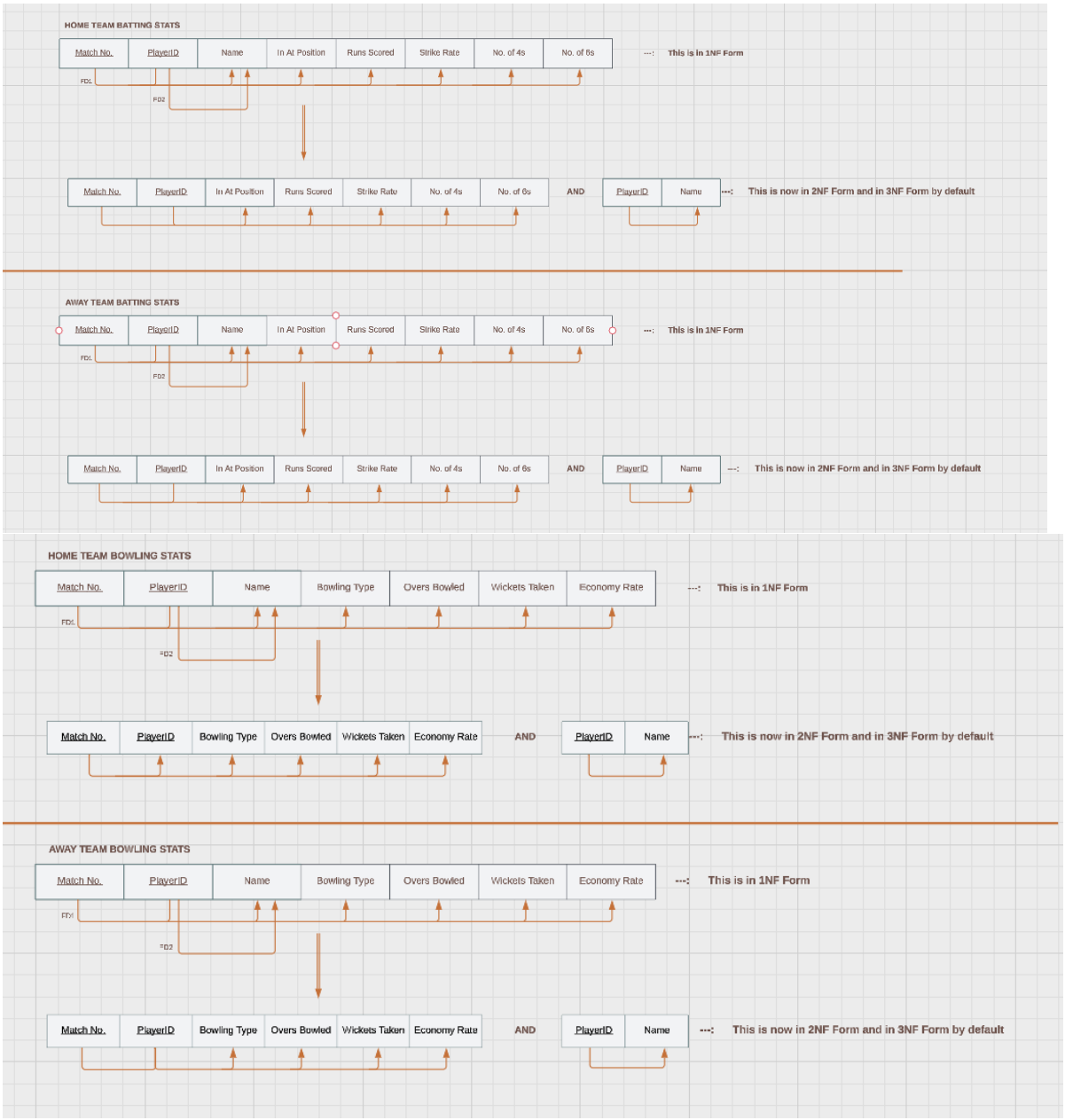


For the Team Stats Tables (such as Home Team Batting Stats):

We are grouping 4 tables under this category and generalizing them since all these 4 tables have a similar structure construct and can be talked about in general.

These four tables have already been split into 2 each during their conversion of 1NF to 2NF form. Let us talk about Home Team Batting Stats table for example. This table has two sub tables, each of which has a single functional dependency. This doesn't create a chance for transitive dependency in any way. Thus, we can conclude that the split tables, which were earlier thought to be in 2NF form are also in 3NF form by virtue of their construction.

Similar things can be said about the Away Team Batting Stats, Home Team Bowling Stats, Away Team Bowling Stats tables as well. Each of these tables have sub-tables in which each has a single functional dependency. This ensures that the above-mentioned tables are all in 3NF form by virtue of their construction.



Thus, we have successfully converted all the tables in our relational schema into 3NF form. We now proceed to the SQL queries part.

e) SQL Queries for User Requests:

Finally, we come to the SQL Queries for the User Requests. First, we must write a code to create the database ‘Cricket Match Management System’ and to create the above-mentioned tables.

Now, we must create Views to show the relationships between two tables. For this, we write the code using joins and show the relationships that exist between the tables. In the output, we get the table which together consists of the columns of two entities.

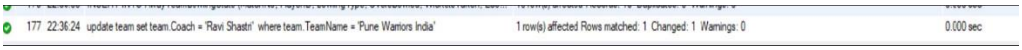
Next job is to populate the tables with sample values to show the queries. For this, we have generated random values corresponding to a single IPL Season, say 2019. We have added around 30 players, 10 teams, showed 10 matches, and the corresponding match officials, and the team stats as well. The output is that the table is populated with all the values that we have entered.

Now, the final part of this portion is to write the SQL Queries for the User Requests. 20 sample queries have been provided to us in the slack channel and we have performed a few necessary changes that suit our database.

Here are the 20 SQL queries that give the output as required. We can see the outputs once we run these queries in the MySQL Workbench. The outputs screenshots are also shown below.

```
-- Query7 : Updating name of coach of TeamName 'Pune Warriors India' to 'Ravi Shastri';

    update team set team.Coach = 'Ravi Shastri'
    where team.TeamName = 'Pune Warriors India';
```




```
-- Query8: Delete all records from the HomeTeamBattingStats and AwayTeamBattingStats tables where runs_scored is less than 50.

    delete from hometeambattingstats where hometeambattingstats.RunsScored < 50;
    delete from awayteambattingstats where awayteambattingstats.RunsScored < 50;

    -- select * from hometeambattingstats;
    -- select * from awayteambattingstats;
```


```
-- Query9: Insert a new match record with match_date "2019-03-15", ground location "Wankhede Stadium", and competing team Names 'Mumbai Indians' and 'Chennai Super Kings'.
    INSERT INTO Matches (MatchNo, MatchDate, GroundLocation, AwayTeamName, MatchWinner, ManOfTheMatch) VALUES
    (11, '2019-03-15', 'Wankhede Stadium', 'Chennai Super Kings', 'Chennai Super Kings', 'MS Dhoni');
    -- select * from matches;
```



```
-- Query10: Update the MatchWinner of MatchNo 5 to Rajasthan Royals and ManOfTheMatch to 'Sanju Samson'
    update matches set matches.MatchWinner = 'Rajasthan Royals', matches.ManOfTheMatch = 'Sanju Samson' where matches.MatchNo = 5;
    -- select * from matches;
```


```
-- Query11: Retrieve the highest runs scored by a player in a match, in the whole season/of all the matches played till now.
    Select max(RunsScored) as MaxRunsScoredByAPlayerInAMatch from (select RunsScored from HomeTeamBattingStats
    UNION select RunsScored from AwayTeamBattingStats) AS BattingStats;
```

```sql
-- Query12: Retrieve the total number of matches won by each team till now.
Select TeamName,No_Of_Wins from Team;
```

TeamName	No_Of_Wins
Chennai Super Kings	9
Delhi Capitals	9
Gujarat Titans	6
Kings XI Punjab	6
Kolkata Knight Riders	7
Mumbai Indians	11
Pune Warriors India	3
Rajasthan Royals	5
Royal Challengers Bangalore	5

```sql
-- Query13: Retrieve the average age of players in each team.
select TeamName,avg(Age) from playerplaysforteam group by TeamName;
```

TeamName	avg(Age)
Royal Challengers Bangalore	31.0000
Mumbai Indians	27.5000
Kolkata Knight Riders	29.0000
Sunrisers Hyderabad	27.2500
Chennai Super Kings	34.2500
Delhi Capitals	26.6667
Gujarat Titans	40.3333
Kings XI Punjab	28.6667
Rajasthan Royals	25.0000

```sql
-- Query14: Retrieve the highest wickets taken by a player in a match, in the whole season/of all the matches played till now.
Select max(WicketsTaken) as MaxWicketsTakenByAPlayerInAMatch from (select WicketsTaken from hometeambowlingstats
UNION select WicketsTaken from awayteambowlingstats) as BowlingStats;
```

MaxWicketsTakenByAPlayerInAMatch
4

```sql
--    Query15: Retrive the team with the most number of runs scored across all matches
select teamname, max(SumRunsScored) from (select teamname, sum(RunsScored) as SumRunsScored from
(Select teamname,Runsscored from((select HomeTeamName as TeamName, RunsScored from matches_complete
inner join matchhashometeambattingstats on matchhashometeambattingstats.matchno = matches_complete.matchno)
UNION (select AwayTeamName as TeamName, RunsScored as SumOfRuns from matches_complete inner join  matchhasawayteambattingstats on
matchhasawayteambattingstats.matchno = matches_complete.matchno)) as RunSums) as TeamsRunsScored group by teamname) as MaxTeamRunsScored;
```

teamname	max(SumRunsScored)
Rajasthan Royals	160

```sql
-- Query16: Retrive the PlayerName and InAtPosition for Mumbai Indians in different matches.
select TeamName, matchno, Name, InAtPosition from
(( select HomeTeamName as TeamName,matches_complete.matchno,Name_exp_5 as Name,InAtPosition from matches_complete
inner join matchhashometeambattingstats on matchhashometeambattingstats.matchno = matches_complete.matchno)UNION
(select AwayTeamName as TeamName, matches_complete.matchno,Name_exp_5 as Name,InAtPosition from matches_complete
inner join  matchhasawayteambattingstats on matchhasawayteambattingstats.matchno = matches_complete.matchno))
as TotalBattingStats where TeamName = 'Mumbai Indians';
```

TeamName	matchno	Name	InAtPosition
Mumbai Indians	7	Hardik Pandya	6
Mumbai Indians	1	Quinton de Kock	1

```sql
-- Query17 : Retrieve the TeamName, CoachName of the Winning Teams in different matches.
Select matchno,TeamName as WinningTeamName,Coach as WinningTeamCoach from( select matchno,MatchWinner as TeamName, coach from matches
inner join team on matches.MatchWinner = Team.TeamName) as WinningTeamDetails;
```

matchno	WinningTeamName	WinningTeamCoach
1	Mumbai Indians	Mahela Jayawardene
2	Royal Challengers Bangalore	Gary Kirsten
3	Delhi Capitals	Ricky Ponting
4	Chennai Super Kings	Stephen Fleming
5	Rajasthan Royals	Paddy Upton
6	Chennai Super Kings	Stephen Fleming
7	Mumbai Indians	Mahela Jayawardene
8	Sunrisers Hyderabad	Tom Moody
9	Chennai Super Kings	Stephen Fleming

```sql
-- Query18 : Retrive the match_date,location and player name for all the matches played by 'Hardik Pandya'
Select matchdate,groundlocation,name from ((select matchdate,groundlocation,name from matches_complete inner join player
where matches_complete.hometeamname = player.teamname) UNION
(select matchdate,groundlocation,name from matches_complete inner join player where matches_complete.awayteamname = player.teamname))
AS Allplayerdetails where name = 'Hardik Pandya';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

matchdate	groundlocation	name
2019-04-04	Wankhede Stadium	Hardik Pandya
2019-03-15	Wankhede Stadium	Hardik Pandya
2019-03-31	M. A. Chidambaram Stadium	Hardik Pandya

```sql
-- Query19 : Retrieve the team_name,match_date and runs_scored for all matches where a specific player scored more than 50 runs.
select TeamName,MatchDate,RunsScored from ((select HomeTeamName as TeamName,matches_complete.matchdate,RunsScored from matches_complete
inner join matchhashometeambattingstats on matches_complete.matchno = matchhashometeambattingstats.matchno)
union
(select AwayTeamName as TeamName,matches_complete.matchdate,RunsScored from matches_complete
inner join matchhasawayteambattingstats on matches_complete.matchno = matchhasawayteambattingstats.matchno)) as AllBattingStats where RunsScored > 50;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

TeamName	MatchDate	RunsScored
Rajasthan Royals	2019-04-02	78
Chennai Super Kings	2019-04-03	87
Mumbai Indians	2019-04-04	91
Delhi Capitals	2019-04-06	66
Royal Challengers Bangalore	2019-04-07	64
Mumbai Indians	2019-03-31	69
Kolkata Knight Riders	2019-04-01	52
Delhi Capitals	2019-04-01	61
Rajasthan Royals	2019-04-02	62

```sql
-- Query20 : Retrieve player_name, match_date, wickets_taken where no of wickets taken is greater than 2.
select name as PlayerName,matchdate,wicketstaken from ((select name_exp_5 as name,matchdate,wicketstaken from matchhashometeambowlingstats)
union (select name_exp_5 as name,matchdate,wicketstaken from matchhasawayteambowlingstats)) as totalbowlingstats where wicketstaken > 2;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

PlayerName	matchdate	wicketstaken
Deepak Chahar	2019-03-31	3
Rashid Khan	2019-04-01	4
Kagiso Rabada	2019-04-06	3